młody TECHNIK

InforMik

MAGAZYN KOMPUTEROWY „MŁODEGO TECHNIKA"

I
1989



Nasz komentarz:

INFO '88

Niektórzy Czytelnicy mieli mi za złe niewczesny entuzjazm, z którym w pierwszym numerze „InforMika" stwierdzałem koniunkturę dla know-how. Wydaje się jednak, że przynajmniej częściowo miałem rację, gdyż dziś prestiż zawodów „komputerowych" rośnie powoli, ale wyraźnie. Osiągnęliśmy już nawet stan, w którym praca dobrego programisty jest honorowana wyżej niż boiskowe wysiłki drugoligowego kopacza piłki, zwanego u nas kurtuazyjnie piłkarzem. Sytuacja taka zachodzi zwłaszcza wtedy, gdy programista jest zatrudniony w firmie prywatnej, gdyż tam bez wytycznych, wskaźników, tabel, metod normowania pracy itd. udaje się jednak na ogół dość wymiernie powiązać pensję z osobistą efektywnością. Mówię celowo: efektywnością a nie pracowitością, gdyż osobiście nie uważam pracowitości za zaletę samą w sobie. Co innego, gdy przynosi ona oczekiwane efekty, uwaga: nie efekty, ale właśnie oczekiwane efekty. Cóż, przywykliśmy fetyszyzować pracę, rozumianą jako szlachetną okazję do urobienia sobie rąk po łokcie, nie zaś jako konieczny wprawdzie, ale niekoniecznie kochany środek osiągania pewnych celów praktycznych. Mówić w tym kontekście o wynikach toć to przecież zniewaga!

Obserwując burzliwy rozwój firm prywatnych łatwo zauważyć jednak pewne niezbyt budujące zjawisko, a mianowicie wyczerpywanie się naturalnych zasobów odpowiedniej kadry. Pierwsze firmy mogły wybierać personel niczym ulęgałki, wykupując najwartościowsze jednostki z firm państwowych. Następne i mniej zasobne firmy prywatne miały już gorszy wybór, i oto jego efekty. Wchodzę do pomieszczenia pewnej spółki z o.o., szumnie nazwanego biurem techniczno-handlowym. Wystrój wnętrza? Ho, ho! Na półkach okazale prezentują się komputery, drukarki, monitory, kalkulatory. Przejdźmy jednak do konkretów. Pan prezes? Nie ma. Pani kierowniczka? Dokądś wyszła. Pani księgowa? Będzie później. Pan Stasio, spec od komputerów? Chyba chory, ale nie wiadomo na pewno. Na stanowisku dzielnie trwa tylko „zrobiona" na lalę pani sekretarka, a w kącie zza gazety wystaje jeszcze twarz pana Tadka, pałaszującego bułkę z cebulą. Konkurencja nie jest jeszcze dość silna, aby takie firmy odpadały w przedbiegach, plajtując i robiąc miejsce innym, lepszym.

Nowo wprowadzone ułatwienia w tworzeniu firm oznaczają obniżenie poprzeczki energii i determinacji, potrzebnej do założenia firmy. Ta sama energia i determinacja jest jednak niezbędna także do porządnego prowadzenia firmy na dłuższą metę. W firmach prywatnych pracują ci sami ludzie, którzy do niedawna robili „na państwowym". Ich szefowie, świeżo upieczeni przedsiębiorcy, także rekrutują się z byłych pracowników tego sektora. Czy można mówić o wychowywaniu personelu? Wychowanie opiera się na pewnych tradycjach. Tradycje te trzeba niestety tworzyć prawie od podstaw.

Państwowy sektor gospodarki jest niedoskonały, wiemy o tym, ale sektor prywatny także ma swe słabości, z tym, że często zlokalizowane na przeciwnym biegunie. Dobrą okazję do zaobserwowania tego zjawiska dały dwie imprezy informatyczne o profilu targowym, które odbyły się w Katowicach pod koniec roku 1988 w dwumiesięcznym zaledwie odstępie: Softarg '88 i Informacja '88.

Na organizowanym przez cieszący się prestiżem Ośrodek Postępu Technicznego Softargu zjawiła się większość liczących się w Polsce firm komputerowych, w tym wszyscy wielcy potentaci, a impreza była zorganizowana w sposób profesjonalny. Strona marketingowa była natomiast totalną klapą. Słabo i niemrawo reklamowana impreza miała kiepską frekwencję. Nie można tego powiedzieć o Informacji '88. Wręcz przeciwnie. Gromkie bicie w tam-tamy rozlegało się już na długo przed imprezą, w prasie pojawiały się regularne ogłoszenia, ba, o imprezie informowało nawet radio i popularny Teleexpress. W praktyce impreza reklamowana jako międzynarodowe targi okazała się głównie kiermaszem pośredników o dość lokalnym zasięgu. Irytująca była organizacja i merytoryczna nieporadność organizatora, w której to roli wystąpiła katowicka firma PRO-INFO, który nie zdołał np. zmontować sprawnie działającego biura informacyjnego. Poważniejsza próba dowiedzenia się tam czegokolwiek wywoływała konsternację ulokowanych w biurze panienek i gorączkowe (albo i nie) poszukiwania kogoś, kto ewentualnie mógłby coś wiedzieć. Brakowało katalogu targowego, a w folderze, który stanowił jego namiastkę brakowało zarówno części uczestniczących firm, jak i występowały firmy nie biorące udziału w targach. Wyraźnie brakowało firm nadających ton na rynku, jak np. InterAms lub CSK. Nie wykluczone, że przyczyną tego była dość szczególna opinia organizatora, będącego spadkobiercą osławionego studia „JACKIE". Spadkobierca okazał się godny wspaniałych tradycji, czego dowodzi następujący incydent. Oto z megafonów sączy się słodki głos spikera, zapewniającego, iż PRO-INFO sprzedaje wyłącznie własne opracowania itd. Kilka kroków dalej, na stoisku, PRO-INFO jak gdyby nigdy nic rozdaje ulotki, w których podaje za własne opracowania stworzone przez kogo innego i to bynajmniej nie dla niej oraz sprzedaje kopie cudzych opracowań literaturowych, naturalnie bez zgody ich autora. Inny, typowo europejski akcent: ogłoszono konkurs na program i ufundowano nagrody. Warunkiem zgłoszenia do konkursu była jednak opłata w wysokości 50 tys. zł, a próby dowiedzenia się czegoś bliższego o komisji konkursowej itd. prowadziły tylko do straty czasu. Posądzenie organizatora o zupełną nieudolność byłoby jednak krzywdzące, gdyż były przykłady sprawności. Jednym z nich było kasowanie należności za bilety.

Odbudowanie etosu pracy i stworzenie pewnego pozytywnego, obowiązującego standardu działania firm informatycznych jest niestety zadaniem na lata. Ostra, naturalna selekcja rynkowa będzie możliwa dopiero wtedy, gdy zarówno poważne firmy, jak i ich klienci należycie opanują swoje role i wyuczą się właściwych reakcji oraz instynktownego odróżniania ziarna od plew. Jednym słowem, musi ukształtować się pewna kultura. Zanim to nastąpi, rynek komputerowy będzie rajem dla agresywnych naciągaczy, wydrwigroszy i hochsztaplerów, kształtujących swój obraz nie za pomocą merytorycznych osiągnięć, ale przez natarczywą, często pomysłową i zakrojoną z rozmachem, ale przeważnie fałszywą reklamę.

Roland Wacławek

„Młody Technik — InforMik" wydaje Instytut Wydawniczy „Nasza Księgarnia"

Rada Redakcyjna: doc. dr Zygmunt Dąbrowski, inż. Jerzy Jasiuk, dr Zygmunt Kalisz, mgr Zbigniew Słowiński, mgr inż. Jerzy Siek, dr Zbigniew Płochocki, Piotr Postawka, mgr inż. Roland Wacławek, prof. dr hab. Andrzej K. Wróblewski (przewodniczący), mgr inż. Grzegorz Zalot.

Zespół redakcyjny: *„InforMik"* redaguje zespół *„Młodego Technika"* — Jerzy Klawiński (sekretarz red.), Jacek Nowicki (red.), Dariusz A. Przygoda (red.), Lidia Sadowska-Szlaga (korekta), Józef Trzionka (redaktor naczelny), Roland Wacławek (software), Grzegorz Zalot (hardware), Izabella Żur (red. tech.).

Stali współpracownicy: Wojciech Apel, Tadeusz Basista, Jacek Jędrzejowski, Piotr Postawka, Marek Szczepański, Krzysztof Wiśniewski.

Adres redakcji: ul. Spasowskiego 4, 00-389 Warszawa, lub skr. poczt. 380, 00-950 Warszawa. **Telefony:** centrala: 26-24-31 do 36. Dział Łączności z Czytelnikami — wewn. 60, pozostałe działy: wewn. 42 i 47. Redaktor naczelny: 26-26-27 lub wewn. 87.

Warunki prenumeraty: ogólnie obowiązujące w kraju.

Druk: Zakłady Graficzne w Katowicach. Zam. 0084/4333/9 A-52
Nakład 60 000 + 315 egz.

młody TECHNIK

InforMik

MAGAZYN KOMPUTEROWY
„MŁODEGO TECHNIKA"
NR I [9] ROCZNIK III

SPIS TREŚCI

Zdjęcia w numerze: Władysław P. Jabłoński, Marian Łazarewicz, Tadeusz Rzepecki, Grzegorz Zalot, „BYTE", „Nad Renem".
Numer ilustrowali: Jerzy Flisak, Roman Gaik.



Lubimy je, nawet bardzo... no, może nie wszyscy i nie z tych samych powodów, ale lubimy. Nastolatki, bo jest pretekst do urwania się ze szkoły i poszalenia w efektownym, choć płytkim bajorku wymieszanej nowoczesnej techniki, komercji i pseudozachodniego blichtru. No i te prospekty... Przedstawiciele różnych firm, bo można poszwendać się poza biurem bez nudnego dłubania w dokumentach i wysłuchiwania połajanek „z góry" za niską efektywność poczynań, no a gdyby udało się jeszcze coś sprzedać... Specjaliści rozmaitych zakładów lubią wystawy z przyczyn podobnych, ale nie dokładnie takich samych. A z kolei rozmaitego autoramentu żurnaliści lubią wystawy, bo jest gdzie zarobić parę groszy na reportażyku, wypić kawkę lub koniaczek na poczet kosztów reprezentacyjnych jakiejś firmy, a może nawet dostać parę reklamowych „wziątków"...

Nie cierpię wystaw!!! Czasem odnoszę wrażenie, że niektórzy organizatorzy i wystawcy — zwłaszcza w branży komputerowej — uważają zwiedzających za półgłówków, a to obraża moje (zapewne zbyt wygórowane!) poczucie własnej wartości, opierające się głównie na zdolności w miarę ***logicznego*** *myślenia, bo za fachowca to ja się raczej nie uważam. Otóż sprytni spece od marketingu uważają zapewne, iż i tak 99 procent zwiedzających nic z tego co widzi nie rozumie. Oczywiście takiemu plebsowi wystarczy kilka kolorowych migających planszy, efektowny screen na monitorach i urocze młode damy na stoiskach...*

Interesujące jest też zachowanie obsługi stoisk w czasie wystaw. Te słabsze firmy reklamują się hasłem niemal: ***„Tylko my i IBM!"****. Z kolei ci najlepsi albo nie interesują się prawie nikim, bo wiadomo, że Polaków na najlepszy (a więc i najdroższy) sprzęt nie stać, albo po prostu — jak to się ładnie pisze — zaznaczają swoją obecność na naszym rynku poprzez reklamę wizualną, a stoisko zamykają na klucz i idą w odwiedziny do innych zaprzyjaźnionych firm.*

A najgorzej to jak na wystawę przyjdzie zblazowany fachowiec niezłej klasy w towarzystwie dociekliwego żurnalisty o kompleksie tropiciela ***autentycznych*** *nowości, a nie „farbowanych lisów" wożonych z jednej komputerowej wystawy na drugą. Ci to dopiero narozrabiają! Do zamkniętych wrót dobrych firm walą z pianą na ustach, a po co? Nie dość, że to zwraca uwagę (są to więc osobnicy niedelikatni!), to jeszcze i tak nic nie kupią i zawracają* ***handlowcom*** *głowę pytaniami* ***technicznymi****. Skandal!! W gorszych firmach też się takich nie znosi! Przyjdzie jeden z drugim i zacznie wydziwiać, a i niejeden wstydliwy szczegół publicznie (cham jeden!) ujawni, a na podtykane skwapliwie reklamowe długopisy, zapalniczki czy torby nawet nie spojrzy. Nie chce nawet koniaczku, a i zaproszenie na puchatkowe „co nieco" też (prostak!) zignoruje. Po prostu swołocz!!!*

Nie lubię wystaw... Nie pożywi się tu ktoś, dla kogo najważniejsza jest rzetelna, fachowa, szybka informacja techniczna. A już łowca nowości może sobie nasze wystawy spokojnie darować, zwłaszcza ich część sprzętową. Taki komputerowy polski McDonald nie wabi mnie. Może dlatego, że nie lubię też hamburgerów?

JERZY KLAWIŃSKI



EDWARD KRAWCZYŃSKI
ROMAN KULA

ZASADY EDYCJI I ORGANIZACJI EKRANÓW

W serii kilku artykułów planujemy przedstawić podstawy wykorzystania systemu MSX. Skoncentrujemy się głównie na języku MSX-BASIC i wykorzystaniu podstawowych systemów operacyjnych.

Kilka znanych firm japońskich w porozumieniu z firmą software'ową MICROSOFT opracowało wspólny standard mikrokomputera. Na uwagę zasługuje opracowanie rozbudowanej wersji języka zwanej MSX-BASIC (*MicroSoft eXtended basic*). MSX gwarantuje pełną wymienialność oprogramowania między komputerami różnych firm. Stąd konieczność standaryzacji podstawowych elementów konstrukcji. Zdecydowano się na mikroprocesor Z80, kontroler graficzny Texas Instruments TI 9918A i układ generowania dźwięku General Instruments AY-3-8910.

CHARAKTERYSTYKA KOMPUTERÓW SYSTEMU MSX

Jak wspomniano wyżej firmy japońskie oraz zachodnioeuropejskie wprowadziły lub wprowadzają cały szereg komputerów systemu MSX lub MSX-2. W Polsce pierwszymi komputerami tego systemu sprowadzanymi powszechnie były i są „SPECTRAVIDEO 738" oraz „PANASONIC 2700". W naszych rozważaniach skoncentrujemy się głównie na pierwszym z nich, sprowadzanym przez CSH za pośrednictwem firmy „PROSYSTEM-VIENNA".

Jednostka centralna komputera MSX zawiera procesor Z80. Posiada pamięć dynamiczną RAM powyżej 64 kB, VRAM 16 kB zaś ROM minimum 32 kB.

Do standardowych gniazd komputera MSX należą:
- złącze szeregowe RS232 z możliwością programowania szybkości transmisji, możliwością podłączenia modemów itp.,
- złącze równoległe drukarki systemu MSX,
- złącze dodatkowej stacji dysków,
- gniazda audio-, video- i TV,
- gniazdo magnetofonu,
- gniazda manipulatorów.

Dodatkowym złączem jest pięćdziesięciostykowe gniazdo rozszerzające ROM, podobnie jak w znanym „ATA-

RI 800XL" i pośrednio „AMSTRAD-SCHNEIDER CPC464". Złącze to pozwala na zastosowanie dodatkowych kart umożliwiających:

— rozszerzenie pamięci RAM,
— łączenie i komunikację komputerów w sieci lokalnej,
— wykorzystywanie oprogramowania zawartego w kasetach typu cartridge.

Klawiatura komputera MSX zawiera od 70 do 80 klawiszy w tym: klawisze alfanumeryczne, klawisze ruchu kursora, klawisze funkcyjne i klawisze specjalne.

Układ znaków na klawiaturze jak w maszynie do pisania. Klawisze alfanumeryczne umożliwiają wygenerowanie małych liter alfabetu łacińskiego i cyfr, a wraz z SHIFT lub CAPS LOCK dużych liter oraz znaków takich jak: %, $, –, + itp. W kolejnych trybach pracy klawiatury (GRAPH, SHIFT+GRAPH, CODE SHIFT+CODE) osiągalne są: znaki alfabetu greckiego; szczególne znaki alfabetów niemieckiego i hiszpańskiego; pospolite znaki stosowane w matematyce, muzyce, chemii, a także grach karcianych i szereg znaków geometrycznych i graficznych. Klawisze kierunkowe umożliwiają ruch kursora w czterech podstawowych kierunkach, a także pod kątem 45°, np. lewo-góra. Klawisze funkcyjne pozwalają jednym naciśnięciem wprowadzić całą instrukcję lub po ich przedefiniowaniu dowolnego zestawu o długości do 15 znaków. Klawisze specjalne umożliwiają sterowanie przebiegiem programu i edycją ekranową.

Komputer MSX może pracować w 3—4 trybach ekranowych:

— tekstowych; 40 × 24 znaki, 32 × 24 znaki lub 80 × 24 znaki,
— graficznych; wysokiej rozdzielczości 256 × 192 punkty; obniżonej rozdzielczości 64 × 48 pól.

W trybie graficznym możliwe jest definiowanie do 32 obiektów graficznych (ang. *sprites*) o rozmiarze 8 × 8 lub 16 × 16 punktów oraz wielkości normalnej i powiększonej.

Dźwięk generowany przez komputer może mieć trzy niezależne kanały o zakresie do 8 oktaw.

Komputer posiada wewnętrzną stację dysków obsługiwaną przez system MSX-DOS zajmujący razem z MSX DISC BASIC 16 kB. Daje to możliwość przechowywania programów jak i danych. Możliwe jest również korzystanie z bogatego oprogramowania pracującego pod kontrolą systemu CP/M (oczywiście po wprowadzeniu systemu operacyjnego).

EDYCJA I JEJ ZLECENIA

System MSX umożliwia stosowanie tzw. edytora pełnoekranowego, co oznacza, że możliwe jest dokonanie zmian w treści dowolnej linii programowej zawartej w obrębie ekranu bez konieczności odwoływania się do jej treści jak jest to np. w ZX Spectrum. Edytor umożliwia poruszanie kursorem w obrębie ekranu w sposób ciągły i skokowy, kasowanie znaku, linii, części linii oraz całego ekranu, a także dopisywanie znaków wewnątrz, na początku i końcu linii. Daje to możliwość bardzo szybkiej interwencji w program jednak pod warunkiem poznania dość szerokich możliwości edytora. Wykorzystuje on w tym celu zarówno klawisze specjalne jak również kombinację klawiszy specjalnych i alfanumerycznych. Powyższe możliwości zostały przedstawione w tabeli I.

TABELA I

Kombinacja klawiszy	Wykonywana operacja
CLS/HM	przesuwa kursor na pozycję początkową
CLS/HM+SHIFT	jw. dodatkowo oczyszcza ekran
STOP	wstrzymuje bieg programu w BASIC-u do następnego użycia klawisza STOP
STOP+CTRL	wstrzymuje bezwarunkowo bieg programu
DEL	kasuje znak znajdujący się pod kursorem
←	przesuwa kursor w lewo kasując znaki
→	przesuwa kursor o 5 znaków w prawo kasując je
INS	pozwala wstawiać znaki wewnątrz istniejącego tekstu aż do powtórnego użycia
CTRL+B	przesuwa kursor na początek pierwszego słowa
CTRL+C	przerywa instrukcję INPUT
CTRL+E	kasuje znaki do końca linii
CTRL+F	przesuwa kursor na początek następnego słowa
CTRL+G	generuje dźwięk (BELL)
CTRL+H	odpowiada klawiszowi specjalnemu
CTRL+I	dokonuje tabularyzacji co 8 znaków
CTRL+J	dokłada linię
CTRL+K	równoważne CLS/HM
CTRL+L	oczyszcza ekran; równoważne CLS
CTRL+M	równoważne ENTER
CTRL+N	przesuwa kursor w prawo do końca linii
CTRL+R	równoważne INS
CTRL+U	kasuje linię z kursorem
CTRL+\	kursor w prawo
CTRL+]	kursor w lewo
CTRL+ Λ	kursor w górę
CTRL+—	kursor w dół

INFORMACJE OGÓLNE DOTYCZĄCE JĘZYKA

MSX BASIC powstały przez rozszerzenie wersji 4.5 BASIC-a posiada ogólne założenia zbieżne z GW-BASIC i GWH stworzonym na IBM-PC.

Zawiera on 163 instrukcje i funkcje zastrzeżone. Ich nazwy nie mogą być używane jako nazwy zmiennych. Komputer może pracować w trybie bezpośrednim jak i programowym. Linia programowa musi posiadać swój indywidualny numer z zakresu 0—65529, a następnie jedną lub więcej instrukcji języka BASIC o gramatyce zgodnej z zasadami tego języka, oddzielone dwukropkiem oraz zakończone ⟨ENTER⟩. Pojedyncza linia programowa może zawierać do 255 znaków. Warto wiedzieć że edycja w obrębie istniejącego programu lub dopisanie kolejnej linii powoduje wyzerowanie zmiennych programowych.

MSX BASIC pracuje wykorzystując do operacji arytmetycznych system dziesiętny kodowany dwójkowo. Możliwa jest praca komputera z podwójną precyzją obliczeń (14 cyfr znaczących) lub pojedynczą precyzją (6 cyfr znaczących) z wykorzystaniem stałych zmiennoprzecinkowych z zakresu od (10E – 64) do (10E + 63) oraz praca ze zmiennymi całkowitymi. Stałe mogą być również reprezentowane w systemie dwójkowym (&B), ósemkowym (&0) i szesnastkowym (&H).

Standardowo komputer pracuje z podwójną precyzją obliczeń. Programowo można zadeklarować:
— zmienne całkowite, używając
DEF INT, co równoważne jest oznaczeniu %,
— zmienne łańcuchowe, używając
DEF STR, co równoważne jest oznaczeniu $,
— zmienne pojedynczej precyzji, używając
DEF SNG, co równoważne jest oznaczeniu !,
— zmienne podwójnej precyzji, używając
DEF DBL, co równoważne jest oznaczeniu #,
np.
jeżeli zadeklarujemy A = 10/3 to
A% lub DEF INT A będzie równe 3
A! lub DEF SNG A będzie równe 3.33333
A# lub DEF DBL A będzie równe 3.3333333333333

Należy z kolei pamiętać, że nazwa zmiennej może być reprezentowana literą, literami i liczbami lub słowem z wyjątkiem zastrzeżonych. Zmienną łańcuchową reprezentuje jedna lub dwie litery i znak $. Warto wiedzieć, że deklaracja nazwy zmiennej może być wieloliterowa, ale komputer rozróżnia tylko dwa pierwsze znaki (np. zmienne „lala" oraz „lato" są dla niego tożsame).

EKRANY TEKSTOWE

Rozwiązanie określonego problemu realizowanego przez komputer winno być szybko i przejrzyście przedstawione użytkownikowi.

Postać ogólna instrukcji PRINT służącej temu celowi jest następująca:

PRINT (lista wyrażeń oddzielonych separatorami)

Listę wyrażeń mogą stanowić wyrażenia arytmetyczne, wyrażenia łańcuchowe umieszczone w cudzysłowie lub zmienne łańcuchowe. Jako separatory mogą być użyte: przecinek, średnik lub apostrof. Niech poniższy listing będzie przykładem pokazującym możliwości użycia tej instrukcji:

```
10 CLS
20 PRINT"wiersz nr 0"
30 PRINT
40 PRINT"wiersz nr 2";
50 PRINT" dalej wiersz nr 2"
60 PRINT"wiersz","nr";3
```

co w efekcie da na ekranie:
wiersz nr 0
wiersz nr 2 dalej wiersz nr 2
wiersz nr 3
OK

Instrukcję PRINT można zastąpić operatorem „?" tak więc 100?"MSX-BASIC" jest równoważne 100PRINT"MSX-BASIC". W dalszym ciągu omówione zostaną rozszerzenia instrukcji PRINT umożliwiające szerokie wykorzystanie jej do formatowania wydruku na ekranie. W tym celu omówić musimy użycie instrukcji SCREEN. Służy ona do deklaracji ekranów możliwych do osiągnięcia w systemie MSX.

Pełna instrukcja SCREEN ma postać

SCREEN (tryb pracy ekranu), (rozmiar obiektu graficznego), (stan klawiatury), (szybkość transmisji), (typ dołączonej drukarki)

Po włączeniu komputer pracuje w trybie standardowym określonym instrukcją SCREEN 0,0,1,1,0. Rozszyfrowanie tego stanu umożliwią informacje zawarte w tabeli II.

TABELA II

Wartość parametru	Objaśnienie
Parametr nr 1 — tryb pracy	
0	tryb tekstowy 40×24 zn.
1	tryb tekstowy 32×24 zn.
2	tryb wysokiej rozdzielczości 256×192 pkt.
3	tryb obniżonej rozdzielczości
Parametr nr 2 — wielkość obiektu graficznego	
0	8×8 pkt. wielkość normalna
1	8×8 pkt. wielkość powiększona
2	16×16 pkt. wielkość normalna
3	16×16 pkt. wielkość powiększona
Parametr nr 3 — stan klawiatury	
0	testowanie klawiatury wyłączone
1—255	dźwiękowe potwierdzenie naciśnięcia klawisza
Parametr nr 4 — szybkość transmisji z lub do magnetofonu	
1	1200 bodów
2	2400 bodów
Parametr nr 5 — typ drukarki	
0	drukarka MSX lub zgodna
1—255	drukarka innego typu

Oczywiście można programowo zdefiniować inny tryb pracy np. instrukcją SCREEN 2,,1,2 co oznacza włączony tryb wysokiej rozdzielczości, włączone testowanie klawiatury, szybkość transmisji 2400 bodów.

Można również zadeklarować szerokość ekranu w trybie tekstowym za pomocą instrukcji o postaci:

WIDTH parametr

Parametrem instrukcji jest liczba 0—39 dla SCREEN 0 oraz 0—31 dla SCREEN 1. Proponujemy sprawdzić program

```
10 CLS
20 FOR X=10 TO 20
30 WIDTH X
40 PRINT"szerokosc ekranu ";X
50 FOR K=1 TO 100:NEXT K
60 NEXT X
```

Dyskietka systemowa umożliwia pracę z ekranem osiemdziesięciokolumnowym po wprowadzeniu sekwencji BLOAD"BASIC 80.OBJ",R.

Do umiejscowienia wydruku na ekranach tekstowych służy instrukcja o postaci ogólnej

LOCATE kolumna, wiersz (kursor)

gdzie jako kolumny użyć można stałej lub zmiennej z zakresu 0—39, jako wiersza z zakresu 0—21, a kursora 0 gdy wygaszony lub 1 gdy jest wyświetlany na ekranie. Sprawdź:

```
10 CLS
20 LOCATE14,12,0
30 PRINT"Srodek ekranu"
```

Do tabularyzowania wyników można użyć instrukcji TAB o postaci: TAB (A) gdzie A jest stałą lub zmienną

o zakresie 0—255. Instrukcji TAB użyć można wielokrotnie pod jedną instrukcją PRINT np.:

```
10 CLS
20 A=5
30 PRINT"Dzis ";TAB(10);A;TAB(A^2)"wrzesnia"
```

Instrukcja ta jest szczególnie przydatna dla tabularyzowania danych liczbowych i łańcuchów mogących mieć różną ilość znaków np. w działalności finansowo-księgowej.

Inną bardzo użyteczną formą wykorzystania instrukcji wydruku jest: PRINT USING o postaci:

PRINT USING symbol formatowania; lista wyrażeń

Jako symboli formatowania dla zmiennych łańcuchowych można użyć: „!", „\n-spacji\". „&".
Prosimy wpisać

```
10 CLS
20 PRINTUSING"!";"Ola","Karol"
30 PRINTUSING"\ \ ";"Kraków","to","lód"
40 A$="na lewo":B$="na prawo":PRINTUSING"& most ";A$,B$
```

"& ⊔most ⊔"; a$,b$
RUN
OK
Kra to lód
Na lewo most na prawo most

W instrukcji PRINT USING formatować można również dane numeryczne następującymi operatorami zamkniętymi w " ":
"#","+","−","**","$$","**$",",","∧∧∧∧", "%".

Możliwości wykorzystania tych operatorów pokaże następujący program:

```
10 CLS
20 FOR I=1 TO 6
30 READ A
40 PRINTUSING"####.##";A
50 NEXT I
60 DATA 1,12,-123,123.45,12345.1,.345
```

Sens instrukcji z wierszy 20 i 50 wyjaśniony zostanie później.

RUN
1.00
12.00
−123.00
123.45
%12345.10

Znak % przed liczbą oznacza przekroczenie deklarowanego formatu — została ona wypisana z pominięciem obowiązujących reguł.

— Prosimy zmieniać w wierszu 30 symbole formatowania na następujące:
**###.##,$$###.##,###.
.##∧∧∧∧,+###.##

Podane powyżej przykłady użycia operatorów nie wyczerpują wszystkich możliwości użycia PRINT. Możliwe jest użycie zarówno operatorów, instrukcji LOCATE, TAB oraz różnego sposobu formatowania w jednej instrukcji. Może to dać ciekawe wyniki. Warto spróbować.



EKRANY WYSOKIEJ ROZDZIELCZOŚCI

Dotychczasowe nasze rozważania dotyczyły rozmieszczania znaków na ekranie monitora wyłącznie w trybach tekstowych. Instrukcja SCREEN umożliwia wywołanie również ekranów o podwyższonej rozdzielczości. Deklarując SCREEN 2 współpracujemy z ekranem o rozdzielczości 192 × 256 punktów, zaś SCREEN 3 wywołuje tryb wielokolorowy. Wykorzystuje on współrzędne ekranu drugiego (SCREEN 2) jednak jego rozdzielczość jest zmniejszona czterokrotnie w poziomie i pionie. Tak więc po zadeklarowaniu SCREEN 2 lub SCREEN 3 przechodzimy do trybu graficznego z programowo ustawionym punktem początkowym grafiki w lewym górnym rogu ekranu.

Najprostszą operacją w tym trybie pracy ekranu jest możliwość zaświecenia punktu o kolorze k przez użycie:

PSET (współrzędna **x**, współrzędna **y**), kolor **k**

Współrzędne powinny być odpowiednio z zakresu 0 ÷ 255 oraz 0 ÷ 191. Inne wartości są akceptowane przez system (nie są jednak wtedy wykonywane operacje na ekranie).

System MSX pozwala użyć 16 barw kodowanych wartością parametru **k** z zakresu 0 ÷ 15. Kolorom tym odpowiadają następujące wartości:

0 — bezbarwny	1 — czarny
2 — zielony	3 — jasnozielony
4 — ciemnonieb.	5 — jasnonieb.
6 — ciemnoczerw.	7 — siny
8 — czerwony	9 — jasnoczerw.
10 — ciemnożółty	11 — jasnożółty
12 — ciemnozielony	13 — fioletowy
14 — szary	15 — biały

W instrukcji PSET parametr koloru nie musi być deklarowany, a wtedy kolorem obowiązującym pozostaje kolor wcześniej deklarowany, np. zleceniem COLOR.

Dokładnie odwrotną czynność do instrukcji PSET, a więc wygaszanie punktu dokonujemy używając instrukcji PRESET o identycznym formacie jak PSET. Brak deklaracji koloru **k** powoduje przyjęcie przez punkt koloru tła.

Proponujemy sprawdzić następujący przykład:



```
10 SCREEN 2
20 FOR A=0 TO 255
30 PSET(A+1,50),9
40 PRESET(A,50)
50 NEXT A
```

Imituje on ruch jasnoczerwonego punktu po poziomej prostej oddalonej o 50 punktów od górnego krańca ekranu. Proponujemy sprawdzić ten przykład zmieniając 10 linię na 10 SCREEN 3.

System MSX umożliwia również testowanie stanu punktu graficznego na ekranie drugim lub trzecim. Instrukcja

POINT (współrzędna **x**, współrzędna **y**)

przyjmuje wartość kodu koloru **k** punktu o współrzędnych **x**,**y**. Współrzędne te przyjmują wartości identyczne jak instrukcje PSET i PRESET.

Z kolei instrukcja

PAINT (współrzędna **x**, współrzędna **y**), kolor **k**

o parametrach jak powyżej umożliwia wypełnianie, zamalowywanie figur zamkniętych kolorem **k** począwszy od punktu o współrzędnych **x** i **y**. Oczywiście punkt ten musi znaleźć się we wnętrzu figury, zaś kolor musi być taki sam, jakim wykreślona została figura, gdyż w przeciwnym przypadku zakolorowaniu ulegnie cały ekran.

Dalsze możliwości wykorzystania ekranów graficznych daje użycie instrukcji LINE. Standardowo instrukcja ta ma postać:

LINE (początkowa współrzędna **x**, początkowa współrzędna **y**),
— (końcowa współrzędna **x**, końcowa współrzędna **y**), kolor **k**, parametr **p**

Umożliwia ona kreślenie linii prostych (odcinków) zawartych między współrzędnymi początkowymi a końcowymi w kolorze **k**. Instrukcja LINE pozwala również wykreślić prostokąt oparty na przekątnej określonej w instrukcji współrzędnymi początkowymi i końcowymi. W tym celu jako **p** należy użyć oznaczenia B. Możliwe jest również wypełnienie tegoż prostokąta kolorem **k** po użyciu w miejsce parametru **p** oznaczenia BF. Jeszcze szersze możliwości daje nam użycie rozszerzonej wersji instrukcji LINE o postaci:

LINE STEP (x_p,y_p) — STEP (x_d,y_d),k,p

gdzie:

x_p,y_p — oznacza przesunięcie odpowiednio na osi **x** i **y** początkowego punktu kreślenia odcinka lub prostokąta względem punktu ostatnio użytego,

x_d,y_d — oznacza długość kreślonego odcinka lub przekątnej prostokąta.

Z powyższych informacji wynika możliwość użycia w instrukcjach LINE i LINE STEP ujemnych liczb jako parametrów **x** i **y**.

Należy także dodać, że punkt grafiki wysokiej rozdzielczości ma kształt prostokąta, co wynika z proporcji boków części roboczej ekranu jak również ilości punktów

w pionie i poziomie. Stosunek długości boków elementarnego punktu ekranu wynosi około 1,35 i przez taką liczbę należy przemnożyć przyrost współrzędnej **y** względem przyrostu współrzędnej **x**, aby otrzymać na ekranie figurę regularną np. kwadrat.

Powyższe wyjaśnienia przybliży przykład, który proponujemy sprawdzić: dla ekranu drugiego i trzeciego:

```
10 SCREEN 2
20 LINE(50,50)-(100,100),5,BF
30 LINE-(80,80)
40 LINESTEP(-50,50)-(100,100),10,B
50 PAINT (80,120),10
60 A=POINT(100,100)
70 B=POINT(50,50)
75 FOR T=0TO300:NEXTT:REM pauza
80 SCREEN0
90 PRINT"Kolor 1-go prostokata ma kod nr ";B
100 PRINT"Kolor 2-go prostokata ma kod nr ";A
```

Niezwykle użyteczną instrukcją w systemie MSX jest instrukcja DRAW wykorzystująca rozszerzoną wersję grafiki BASIC. Jej postać jest pozornie prosta:

DRAW „wyrażenie łańcuchowe"

Zawartość łańcucha stanowić mogą litery oraz liczby będące elementami kodu makrografiki BASIC. Litery definiują kolejno wykonywane operacje zaś liczby są argumentami tych operacji. Punktem początkowym kreślonej grafiki jest ostatni punkt, do którego wcześniej odwoływał się program. Poszczególne litery zawarte w „Wyrażeniu łańcuchowym" oznaczają:

U — rysuj linię w górę (ang. *up*),
D — rysuj linię w dół (ang. *down*),
R — rysuj linię w prawo (ang. *right*),
L — rysuj linię w lewo (ang. *left*).

Argumentem tych funkcji jest liczba określająca długość linii w umownych jednostkach określonych literą $

$ — to skala grafiki, i tak dla skali 1:1 argument litery $ wynosi 4, stan ten jest standardowy, zaś $ 8 oznacza dwukrotne powiększenie skali grafiki wykonywanej po tej deklaracji.

Możliwe jest również kreślenie wg powyższych zasad linii ukośnych za pomocą:

E — rysuj linię w prawo-górę,
F — rysuj linię w prawo-dół,
G — rysuj linię w lewo-dół,
H — rysuj linię w lewo-górę.

Argumenty tych funkcji mają identyczne cechy jak wyżej. Poza wymienionymi możliwe jest również użycie następujących liter:

A — z argumentem 0 + 3, co pozwala na obrót grafiki o kąt odpowiednio 90°, 180°, 270°, 360° w kierunku przeciwnym do ruchu wskazówek zegara,
B — umożliwia przesunięcie punktu „rysującego" bez kreślenia linii,
C — ustala kolor kreślonych linii zgodnie z kodami kolorów,
M — kreśli linię od punktu początkowego do punktu określonego argumentami tej funkcji,
N — kreśli linię od punktu początkowego do punktu ostatnio użytego — dokonuje więc powrotu do punktu końcowego przedostatniej operacji,
X — pozwala wykonywać cykl operacji graficznych zdefiniowanych łańcuchem alfanumerycznym będącym argumentem tej funkcji.

Prosimy sprawdzić więc, że zapis

```
10 SCREEN 2
20 PSET(50,50)
30 DRAW"d20u10e12g12f12"
40 GOTO40
```

kreśli dużą literę „K", a jeśli dodać do końca łańcucha A1 wykreślona zostanie litera „K w odbiciu lustrzanym.

Sprawdź samodzielnie następujące dwa przykłady:

```
10 SCREEN 2
20 A$="u3nr3u3r4bm+4,6"
30 PSET(100,100),10
40 DRAW A$+"nl2nr2u6g2bm+6,1r5bm+3,3"+A$+"d5l32u18r32d13"
50 GOTO50
```

oraz

```
10 SCREEN 2
20 A$="u50r35d50l35"
30 FORI=1 TO 20 STEP 2
40 PSET(75+I,75+I)
50 DRAW "xa$;"
60 NEXT
70 GOTO 70
```

Przejdźmy do kolejnej instrukcji graficznej o standardowej postaci:

CIRCLE (współrzędna **x** środka, współrzędna **y** środka) promień **r**, kolor **k**, kąt **u**, współczynnik **e**

umożliwiającej kreślenie okręgów, elips oraz ich fragmentów. Współrzędne środka oraz promień muszą być użyte, aby instrukcja CIRCLE została zdefiniowana. Pozostałe parametry mogą być użyte opcjonalnie.

Definiują one:

kolor **k** — kolor określonej figury,
kąty **u**, **w** — tzw. kąty początkowy i końcowy wyrażane w radianach,
współczynnik **e** — wzajemny stosunek osi dużej i małej dla elipsy.

Prosimy sprawdzić, że

CIRCLE (120,100),50,5,,,1.35

da w efekcie figurę zbliżoną do okręgu, natomiast

CIRCLE (120,100),50,5

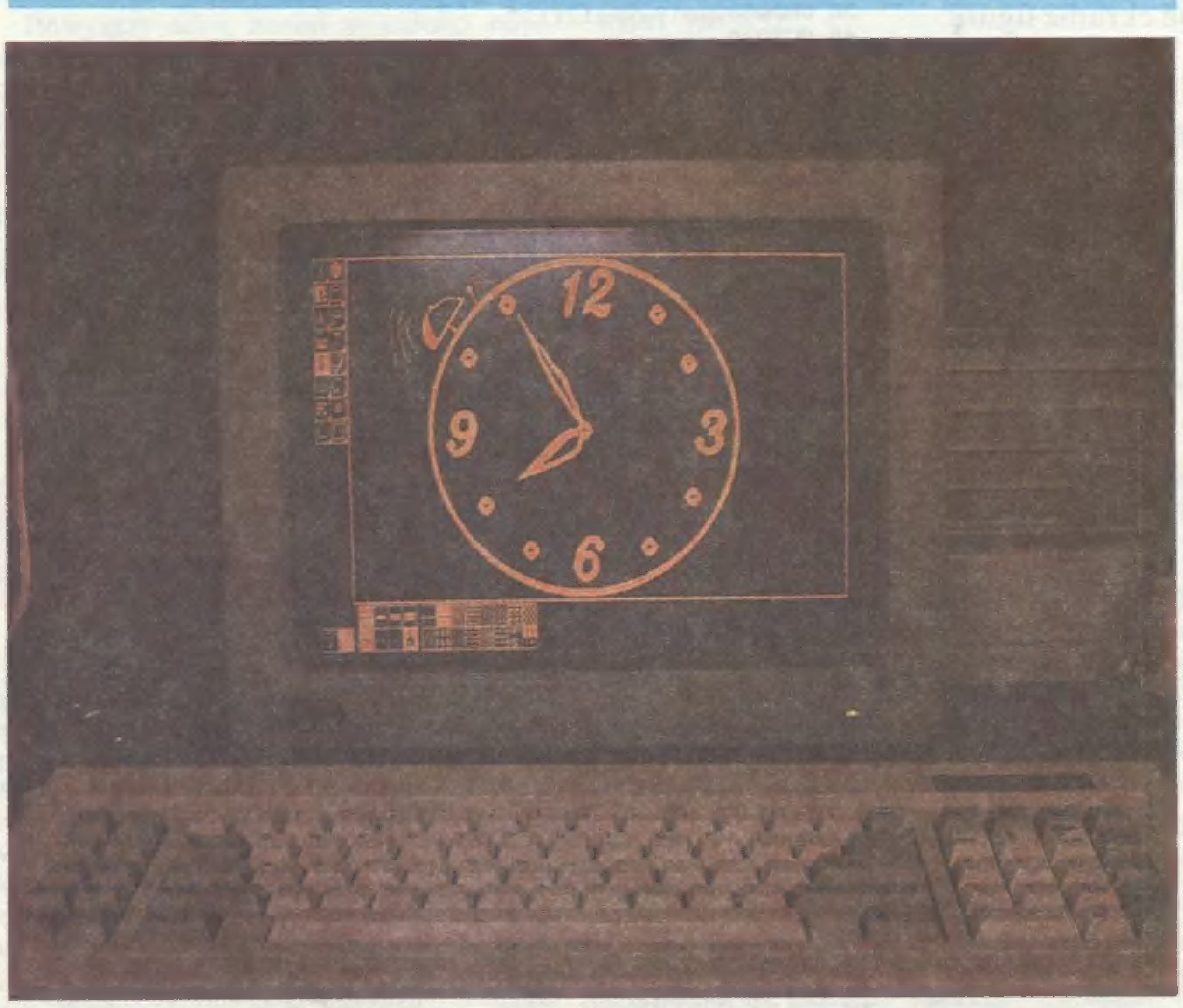
wykreśli elipsę. Wynika to z wcześniej omawianego kształtu elementarnego punktu ekranu.

Przedstawione powyżej problemy edycji ekranowej oraz współpracy z ekranami nie wyczerpują zagadnienia w całości. Naszym zdaniem zamieściliśmy w artykule szereg informacji nieosiągalnych w dostępnej literaturze. W kolejnych publikacjach postaramy się o podobne podejście do dalszych zagadnień dotyczących mikrokomputerów MSX, a na razie proponujemy zainteresowanym tematem samodzielne rozszerzenie informacji drogą studiów następujących pozycji literatury:

MSX SPECTRAVIDEO-SVI 738 — PTE Zielona Góra 1987,

MSX-BASIC Referenz Buch für PANASONIC 2700 CF — Matsushita Electronic Trading Central — Osaka 1986,

MSX BASIC Reference Manual — Spectravideo International Ltd. — 1985.



ROLAND WACŁAWEK

BUDZIK bez trybików i sprężynek

Siedząc przy klawiaturze komputera i oddając się różnym pasjonującym zajęciom bardzo łatwo zapomnieć o upływie czasu, nieprawdaż? Zegar tymczasem tyka miarowo i wnet okazuje się, że przegapiliśmy umówione spotkanie albo audycję telewizyjną (ja stale zapominam o moim ulubionym „Teleexpresie"). Niektóre programy pozwalają wmontować np. w prawy górny róg ekranu zegar cyfrowy. W praktyce jego przydatność okazuje się jednak problematyczna, gdyż szybko oswajamy się z obecnością zegara i przestajemy go dostrzegać. Żeby spojrzeć na zegar, trzeba by pamiętać, po co... Jednym słowem, zamiast zegara przydałby nam się komputerowy budzik, przypominający natarczywym dźwiękiem o zbliżaniu się zaprogramowanej godziny. W niniejszym artykule zajmiemy się majstrowaniem takiego budzika dla komputerów klasy PC/XT i AT, a przy okazji rozważymy sposoby korzystania z przerwań zegarowych w komputerach tej klasy oraz metody rozpoznawania obecności programów rezydujących i przekazywania parametrów do nich.

Podstawowym narzędziem do odmierzania czasu jest w PC/XT i AT układ czasowy 8253. Jeden z jego trzech liczników pracuje jako sterowany kwarcowym rezonatorem dzielnik częstotliwości, dostarczający na wyjściu impulsów o okresie 0.0549254 s, co odpowiada częstotliwości 18.2065 Hz. Każdy z tych impulsów wywołuje przerwanie maskowania IRQ nr 8. Przerwanie to jest normalnie obsługiwane przez procedury zawarte w BIOS, które m.in. zliczają te impulsy zegarowe. Cały ten proces odbywa się naturalnie nieprzerwanie i równolegle z realizacją programów aplikacyjnych. Czterobajtowy licznik zlokalizowany pod adresem absolutnym 0040H:006CH zawiera liczbę impulsów zegarowych, zarejestrowanych od godziny 0:00.

Gdyby przejąć wektor przerwania nr 8, mielibyśmy możliwość wywołania naszego własnego podprogramu co 1/18 sekundy. Właściwość ta byłaby bardzo przydatna także dla budzika, który za każdym razem mógłby sprawdzić, czy przypadkiem nie nadeszła wyznaczona pora. Projektanci firmy IBM przewidzieli taką potrzebę i zaprojektowali dodatkowe przerwanie nr 28 (1CH), tym razem programowe, które jest wywoływane przez BIOS bezpośrednio z procedury obsługi przerwania nr 8. Normalnie wektor przerwania 28 wskazuje na rozkaz **IRET**, zawarty w pamięci BIOS, tak że przerwanie to jest nie wykorzystane. W razie potrzeby można jednak wektor ten „przestawić" na własną procedurę obsługi przerwania zegarowego. Takie rozwiązanie ma dwie zalety: po pierwsze, nie trzeba się specjalnie troszczyć o przekazanie sterowania po zakończeniu procedury obsługi z powrotem do BIOS i zapamiętywać w tym celu adresu procedury standardowej — wystarczy po prostu zakończyć

procedurę rozkazem maszynowym **IRET**; po drugie, standardowa procedura obsługi przerwania zewnętrznego i procedura użytkownika są elegancko rozdzielone, co zmniejsza ryzyko ich wzajemnego, niekontrolowanego oddziaływania na siebie i zmniejsza groźbę popełnienia błędów.

Chcąc odczytać licznik impulsów zegarowych, można to zrobić wprost. Trzeba jednak zadbać, aby w trakcie odczytu przerwania były zablokowane. Inaczej grożą nieprzewidziane efekty. Przypuśćmy, że młodsze słowo (2 bajty) licznika ma wartość 0FFFFH, słowo starsze — 1, co odpowiada dziesiętnej liczbie 131071. Odczytujemy licznik do rejestrów **CX** i **DX** np. poniższą kombinacją rozkazów:

```
MOV  AX, 0040H
MOV  DS, AX
MOV  DX, [006CH]
MOV  CX, [006EH]
```

Pierwsze dwa rozkazy zapewniają tylko właściwy adres segmentowy w rejestrze **DS**. Przypuśćmy, że w chwili zakończenia trzeciego rozkazu wystąpiło przerwanie nr 8. W wyniku kolejnej inkrementacji licznika nastąpiło przeniesienie do słowa starszego, młodsze słowo zostało wyzerowane, a starsze zawiera teraz 2. Licznik zawiera więc liczbę 131072. W rejestrze **DX** znajduje się jednak odczytana wcześniej wartość młodszego słowa, czyli 0FFFFH (65535), natomiast rejestr **CX** zostanie załadowany dopiero po zakończeniu obsługi przerwania i dlatego otrzyma nową wartość starszego słowa licznika, tzn. 2. Odczytana wartość 2*65536+65535 = 196607 jest więc obarczona grubym błędem. Aby uniknąć takiej niespodzianki, należałoby przed odczytem zabronić przerwań (rozkazem **CLI**), a po jego zakończeniu zezwolić na przerwania ponownie (**STI**), np. tak oto:

```
CLI
MOV  DX, [006CH]
MOV  CX, [006EH]
STI
```

W praktyce zawartość licznika impulsów zegarowych wygodniej jest odczytywać za pomocą wyspecjalizowanej funkcji usługowej, BIOS, dostępnej za pośrednictwem przerwania programowego nr 26 (1AH). Jeżeli w chwili wywołania tego przerwania rejestr **AH** zawierał 0, to po powrocie w rejestrach **CX** i **DX** znajdzie się bieżąca wartość licznika (starsze bajty w **CX**). Jeżeli **AH** zawiera 1, to zawartość rejestrów **CX** i **DX** zostanie wpisana do wewnętrznego licznika, określając jego zawartość początkową. Funkcji tej można zatem użyć do „nastawiania" wewnętrznego zegara programowego.

Nasz przyszły budzik po jednorazowym „nastawieniu" powinien stale czuwać i odliczać czas, niezależnie od tego, jaki program aplikacyjny eksploatujemy w danej chwili. Program budzika musi zawierać zatem część rezydującą — instalowaną na stałe w pamięci i odpowiedzialną za bieżące porównywanie zaprogramowanego czasu alarmu z bieżącym czasem systemowym i w razie potrzeby generującą sygnał dźwiękowy. Natomiast funkcje związane z „nastawianiem" budzika może realizować nierezydująca część programu, usuwana z pamięci po zakończeniu pracy.

Część rezydująca przechwyci wektor nr 1CH i „nastawi" go na punkt wejścia własnej procedury obsługi. Procedura obsługi mogłaby kończyć się rozkazem **IRET**. Rozwiązanie to, choć możliwe i bezpieczne, ma jednak pewną wadę. Otóż nasz program budzika może nie być

```
;****************** [illegible] wersja 1.01 *******************
;********** Roland Wacławek,  Siemianowice Śl. 1987 **********

ZegarBIOS EQU  1AH        ;nr przerwania obsługi zegara BIOS
UslugiDOS EQU  21H        ;nr przerwania wywołania usług DOS
PowrotDOS EQU  20H        ;nr przerwania zakończenia programu
Pozostaw  EQU  27H        ;nr fn. czyniącej program rezdującym
LadujWekt EQU  35H        ;numer funkcji odczytującej wektory
[illegible] EQU 25H       ;numer funkcji zmieniającej wektory
Zameldúj  EQU  [illegible] ;nr funkcji PC-DOS emitującej napisy

Kod_prog  SEGMENT PARA
          [illegible] CS:Kod_prog
          [illegible] 129
Parametry LABEL [illegible] ;bufor parametrów wywołania programu

          ORG  256
Start:    JMP  Inicjuj             ;skok do części inicjacyjnej

Haslo:    DB   'R.WAC0288'         ;identyfikator części rezyd.
Minuty:   DB   0                   ;czas alarmu - pamięć minut
Godziny:  [illegible] 0            ;czas alarmu - pamięć godzin
Fl_Aktywn:DB   0                   ;semafor zabraniający obsługi
FL_Sygnal:DB   0                   ;semafor generacji dźwięku
WektorOfs:DW   0                   ;adres starego wektora prze-
WektorSeg:DW   [illegible]         ;rwania zegarowego nr 1CH
Licznik:  DB   18                  ;licznik ignorowanych cykli

Budzik:   TEST BYTE [illegible] FL_Sygnal,0FFH  ;trwa emisja dźwięku?
          JNZ  Dzwiek              ;tak-skok do obsługi dźwięku
          TEST [illegible] PTR Fl_Aktywn,[illegible] ;wejście zabronione?
          JE   [illegible]         ;jeśli nie, to obsłuż budzik
E0:       [illegible] Koniec       ;tak - pomiń obsługę budzika
          DEC  [illegible] PTR Licznik  ;odlicz kolejną 1/18 sekundy
          JNE  E0                  ;skok, gdy licznik niezerowy
          MOV  BYTE [illegible] Fl_Aktywn,1 ;zabroń powtórnego wejścia
          STI                 ;odblokuj przerwania maskowalne
          PUSH [illegible]    ;przechowaj rejestry [illegible] stosie
          [illegible] CX
          PUSH [illegible]
          [illegible] AH,AH        ;zeruj rejestr AH - funkcja nr 0
          [illegible] ZegarBIOS    ;odczytaj czas z BIOS do CX:DX
          [illegible] AX,DX        ;przenieś liczbę rejestrowanych
          [illegible] DX,CX        ;impulsów z pary CX:DX do DX:AX
          ADD  AX,AX               ;pomnóż liczbę zawartą w parze
          [illegible] DX,[illegible] ;rejestrów DX:AX przez [illegible] drogą
          [illegible] AX,AX        ;trzykrotnego dodawania jej do
          ADC  DX,[illegible]      ;samej siebie
          [illegible] AX,[illegible]
          ADC  DX,[illegible]      ;DX:AX zawiera liczbę cykli x [illegible]
          [illegible] CX,8739      ;dzielnik (8739.12=18.2065*60*8)
          DIV  CX                  ;AX=liczba minut od godziny 0:00
          [illegible] CL,60        ;podziel liczbę minut przez 60
          DIV  CL                  ;AH=minuty czasu akt., AL=godziny
          [illegible] AH,AL        ;minuty do rej. AL, godziny do AH
          [illegible] AX,WORD [illegible] Minuty ;porównaj czas akt. i alarmu
          JC   BezAlarmu           ,skok,jeśli czas alarmu>czas akt.
          [illegible] [illegible] [illegible] FL_Sygnal,1 ;ustaw semafor gen. tonu
          [illegible] [illegible] PTR Licznik,18*6 ;ładuj licznik cykli tonu
```

jedynym, wykorzystującym wektor 1CH i przerwania zegarowe. Przypuśćmy, że wcześniej zainstalowaliśmy w pamięci np. program zwykłego zegara cyfrowego, który przejął przerwanie 1CH. Następnie ładujemy nasz budzik, który także przejmuje to przerwanie, „kradnąc" je zegarowi, który od tej pory beznadziejnie stoi. Aby tego uniknąć, zamiast rozkazem **IRET** zakończymy procedurę obsługi rozkazem skoku do starego adresu (musimy go oczywiście wcześniej zapamiętać). Jeżeli wektor ten wskazywał na rozkaz **IRET** w BIOS (stan wyjściowy), to nic się nie zmieni, w przeciwnym razie zostanie wykonana wcześniej zainstalowana procedura obsługi, zupełnie tak samo, jak gdyby skok do niej nastąpił od razu po przerwaniu.

Niech gotowy program nosi nazwę BUDZIK.COM. Jego wywołanie będzie miało postać:

BUDZIK gg:mm

gdzie **gg** i **mm** to grupy złożone z 1 lub 2 cyfr i rozdzielone dwukropkiem, a przedstawiające odpowiednio godzinę i minutę alarmu, np. BUDZIK 12:53 albo BUDZIK 7:0. Aby wyłączyć budzik, wystarczy podać czas spoza sensownego zakresu, np. 25:0, albo użyć zlecenia BUDZIK bez argumentów.

Przy pierwszym wywołaniu rezydująca część programu pozostanie w pamięci, ale przy każdym następnym wywołaniu instalowanie programu w pamięci nie powinno już mieć miejsca. Wystarczy zmienić zapisany w części rezydującej czas alarmu. Program musi więc potrafić stwierdzić istnienie w pamięci swej części rezydującej, a w razie jej wykrycia — zrezygnować z ponownej instalacji.

Jak stwierdzić obecność programu BUDZIK w pamięci? Bardzo prosto. W ustalonym miejscu części rezydującej wpiszemy pewną charakterystyczną sekwencję bajtów, stanowiącą „hasło". Po każdym wywołaniu programu BUDZIK najpierw odczytamy bieżący wektor 1CH. Teraz wystarczy porównać zapisane w programie BUDZIK.COM hasło z domniemanym fragmentem pamięci, w którym powinno znajdować się hasło w programie rezydującym. Jeżeli program jest już w pamięci, to segment tego wektora jest równocześnie adresem segmentowym części rezydującej, a przesunięcia adresowe (*offset*) poszczególnych fragmentów programu są identyczne zarówno w programie BUDZIK.COM dopiero co załadowanym z dysku, jak i w kopii jego fragmentu, rezydującej już w pamięci. Jeżeli porównanie da wynik pozytywny, możemy śmiało przyjąć, że część rezydującą załadowano już wcześniej. W przeciwnym razie część rezydującą trzeba dopiero załadować. Warto zauważyć, że opisana metoda identyfikacji będzie skuteczna tylko wtedy, gdy po programie BUDZIK nie załadowano żadnego innego programu rezydującego, „przeginającego" wektor przerwania nr 1CH!

Niezależnie od tego, czy program BUDZIK już rezyduje, czy też nie, należy przeanalizować parametry zlecenia i zapisać odpowiedni czas alarmu. Parametry zlecenia znajdują się w nagłówku programu od adresu 129, gdzie umieszcza ję system operacyjny. Znak ⟨CR⟩ (ASCII 13 lub 0CH) kończy linię zlecenia. Ewentualne spacje pomijamy, poza tym dozwolone są tylko cyfry i dwukropek. Każdy inny znak przerywa analizę i powoduje wysłanie komunikatu o błędzie. Po rozpoznaniu cyfry dotychczasowa wartość liczby godzin lub minut jest mnożona przez podstawę systemu, tj. przez 10, po czym dodawana jest do niej wartość cyfry. Program sprawdza, czy wy-

```
           CLI                      ;zabron reakcji na przerwania IRQ
           MOV  [illegible] [illegible] FL_Aktywn,00000010B   ;wyłącz budzik
           JMP  Powrot              ;to juz koniec obsługi budzika

Dzwiek:    PUSH AX                  ;przechowaj [illegible] stosie rejestr AX
           DEC  BYTE PTR Licznik    ;odlicz kolejny cykl tonu
           JS   StopTon             ;skok, jezeli licznik juz ujemny
           TEST BYTE PTR Licznik,1  ;czy licznik jest parzysty?
           CLI                      ;zabron reakcji na przerwania IRQ
           JNZ  WlaczTon            ;licznik nieparzysty-włącz dźwiek
           IN   AL,61H              ;licznik parzysty - wyłącz dźwiek
           AND  AL,11111100B        ;przez wyzerowanie bitów nr 0 i 1
           OUT  61H,AL              ;w rejestrze nr 61H
           JMP  KoniecTon           ;opusc procedurę obsługi dźwieku

WlaczTon:  MOV  AL,0B6H             ;zaprogramuj dzielnik czestotliw.
           OUT  43H,AL              ;wpisując bajt ster. do portu 43H
           MOV  AL,[illegible]      ;zapisz dwubajtowy współczynnik
           OUT  42H,AL              ;podziału czestotliwosci do re-
           OUT  42H,AL              ;jestru nr 42H (kolejno 2 bajty)
           IN   AL,61H              ;rozpocznij generację dźwieku
           OR   AL,11B              ;przez ustawienie bitów nr 0 i 1
           OUT  61H,AL              ;w rejestrze nr 61H
           JMP  KoniecTon           ;opuść procedurę obsługi dźwięku
StopTon:   MOV  BYTE [illegible] FL_Sygnal,0  ;zeruj semafor gen. tonu
           JMP  KoniecTon                     ;opuść program generacji

BezAlarmu: MOV  BYTE PTR FL_Aktywn,0  ;umozliw ponowne wejscie
           MOV  BYTE PTR Licznik,18   ;inicjuj ponownie licznik
Powrot:    POP  DX                    ;odtwórz uzywane rejestry
           POP  CX
KoniecTon: POP  AX
Koniec:    JMP  DWORD PTR WektorOfs   ;skok do starej procedury

           ASSUME  DS:Kod_prog
Inicjuj:   MOV  AH, LadujWekt         ;wywołaj funkcję PC-DOS,
           MOV  AL, 1CH               ;ładując wektor przerwa-
           INT  UslugiDOS             ;nia nr 1CH do rej. ES:BX
           MOV  WORD PTR WektorSeg,ES ;pamiętaj segment wektora
           MOV  WORD [illegible] WektorOfs,BX ;pamiętaj offset wektora
           MOV  SI, OFFSET Haslo      ;SI:offset identyfikatora
           MOV  DI, SI                ;DI:offset identyfikatora
           MOV  CX,4                  ;licz. porównywanych słów
           REPE CMPSW               ;porównuj kod i identyfikator
           JE   Rezyduje            ;zgodnosc-program juz rezyduje
           PUSH DS                  ;skopiuj do rejestru [illegible] bie-
           [illegible] ES           ;zącą zawartosc rejestru DS
Rezyduje:  OR   [illegible] [illegible] ES:Fl_Aktywn,10B;czasowo wyłącz budzik
           MOV  SI, OFFSET Parametry ;adres parametrow do SI
           MOV  CH,2                ;załaduj licznik dwukropków
           MOV  CL,3                ;załaduj licznik cyfr godzin
           XOR  AH,AH               ;wyzeruj liczbę godzin [illegible] AH
LadujZnak: LODSB                    ;nast. bajt bl. param. do AL
           CMP  AL, 0DH             ;czy koniec bloku parametrów?
           JE   KoniecPar           ;tak, bo znak <CR> kończy blok
           CMP  AL,' '              ;czy kolejny znak jest spacją?
           JE   LadujZnak           ;jeśli tak, to zignoruj spację
           CMP  AL,':'              ;czy ten znak to dwukropek?
```

stąpił dwukropek i czy liczba cyfr w specyfikacji minut i godzin nie była większa od 2. Jeżeli wartość jest formalnie poprawna, zostanie zapisana w odpowiedniej komórce rezydującej części programu.

Adres segmentowy rezydującej części programu znajduje się w rejestrze ES. Jeżeli program już rezydował, wpisujemy do ES adres segmentowy bieżącego wektora przerwania nr 1CH, w przeciwnym razie po prostu kopiujemy do ES rejestr CS (pozostawi on przecież w pamięci własną część rezydującą). Zgodność ES i CS posłuży zresztą w dalszej części programu za wskaźnik czy instalować część rezydującą.

Przejdźmy teraz do procedury obsługi budzika, stanowiącej rdzeń całego programu. Jest ona wywoływana ok. 18 razy na sekundę. W zasadzie można by po każdym wywołaniu sprawdzać, czy bieżący czas systemowy zgadza się z zaprogramowanym czasem alarmu. Nasz budzik nie musi być jednak lekkoatletycznym stoperem, reakcja z dokładnością do ok. sekundy jest aż nadto wystarczająca. Przyjmijmy więc, że tylko co 18 wywołanie spowoduje badanie zgodności czasu, pozostałe będą pomijane. Takie rozwiązanie zmniejsza „narzut" czasowy wprowadzany do systemu wskutek obecności budzika. Co prawda operacje sprawdzające są w naszym przypadku dość proste, ale w innym zastosowaniu mogłyby być bardziej złożone. Metodą obsługi co któregoś przerwania zegarowego można wyważyć proporcję między szybkością reakcji a narzutem czasowym. Odliczanie co 18 cyklu odbywa się przy udziale zmiennej: **LICZNIK**.

Po osiągnięciu czasu alarmu najprościej byłoby nie opuszczać obsługi przerwania aż do czasu wyemitowania całego sygnału dźwiękowego. Istotny mankament polega na tym, że na ten czas jest zablokowana reakcja na przerwania zegarowe w innych, współbieżnych programach, także korzystających z przerwania nr 1CH, a także wszelka inna aktywność tych programów (komputer czasowo „zamiera"). Zastosujemy inny wariant: ustawimy wskaźnik trybu emisji dźwięku (zmienna: Fl_sygnal), który po każdym kolejnym przerwaniu zegarowym spowoduje przekazanie sterowania zamiast do procedury porównywania czasów — do programu emisji dźwięku. Program ten odlicza kolejne cykle (znowu zmienna: **Licznik**). Jeżeli licznik jest nieparzysty, program włączy emisję dźwięku, w przeciwnym razie — wyłączy go, po czym zakończy obsługę przerwania. Tak więc co 1/18 sekundy dźwięk na przemian jest włączany i wyłączany. Ponieważ emisja dźwięku odbywa się z użyciem układu czasowego 8253, bez absorbowania procesora, przerywany dźwięk o czasie trwania ok. 6 sekund będzie emitowany bez blokowania pracy bieżącego programu.

Samo programowanie generatora dźwięku to już materiał na osobny artykuł. Tutaj wspomnijmy, że trzeba w tym celu wykonać dwie czynności: zaprogramować jeden z liczników układu 8253 do pracy w charakterze dzielnika częstotliwości z odpowiednim współczynnikiem podziału, oraz odblokować wyjście na głośnik, ustawiając dwa najmłodsze bity w rejestrze układu równoległego wejścia-wyjścia 8255. Do zablokowania emisji dźwięku wystarczy wyzerowanie tych bitów.

Jeszcze garść szczegółów. Do czego służy zmienna **Fl_Aktywn?** Ustawienie któregokolwiek z jej bitów uniemożliwia wywołanie procedury obsługi budzika. Może to być przydatne w dwóch przypadkach: wtedy,

```
            JE   Dwukropek         ;jeżeli tak,  przejdź do minut
            CMP  AL,'9'            ;kod znaku  większy niż kod 9?
            JA   BlednyPar         ;jeśli tak,podano zły parametr
            [illegible] AL,'0'     ;AL:=  kod znaku - kod cyfry 0
            JS   BlednyPar         ;różnica ujemna = zły parametr
            [illegible] CL         ;odlicz kolejną cyfrę w grupie
            JZ   BlednyPar         ;jeżeli liczba cyfr>2, to błąd
            ADD  AH,AH             ;pomnóż bieżącą wartość liczby
            ADD  AL,AH             ;przez 10,dodając do niej przy
            ADD  AH,AH             ;okazji wartość ostatnio wpro-
            ADD  AH,[illegible]    ;wadzonej cyfry
            ADD  AH,AL
            JMP  LadujZnak         ;analizuj  kolejny znak param.

Dwukropek: DEC  CH                 ;czy  to  pierwszy  dwukropek?
            JNZ  PiszGodz          ;tak - pamiętaj godzinę alarmu
BlednyPar: MOV  DX,OFFSET MeldBlPar ;DX:=adres melunku [illegible] błędzie
Bezinstal: MOV  AH,Zamelduj        ;wywołaj funkcję PC-DOS,emitu-
            INT  UslugiDOS         ;jącą meldunki  o adresie [illegible] DX
            INT  PowrotDOS         ;zwyczajny  powrót  do  PC-DOS

PiszGodz:  MOV  BYTE PTR ES:Godziny,AH   ;zapisz godzinę alarmu
            XOR  AH,AH             ;wyzeruj  liczbę  minut  [illegible] AH
            MOV  CL,3              ;załaduj  licznik  cyfr  minut
            JMP  LadujZnak         ;analizuj  kolejny znak param.
KoniecPar: DEC  CH                 ;czy wystąpił  juz  dwukropek?
            JNZ  BlednyPar         ;nie- nie podano minut - błąd!
            MOV  BYTE PTR ES:Minuty,AH    ;zapisz  minutę alarmu
            MOV  BYTE PTR ES:Licznik,18   ;inicjuj licznik cykli
            AND  BYTE [illegible] ES:FL_Aktywn,11111101B
            MOV  AX,ES             ;czy DS<>ES,co znaczy, [illegible] pro-
            CMP  AX,WORD PTR WektorSeg;gram jeszcze nie rezuduje
            MOV  DX,OFFSET MeldNast;DX:=adres meldunku o nastawie
            JE   BezInstal         ;skocz,  jeżeli  już  rezyduje
            MOV  DX,OFFSET MeldInst;DX:=adres meldunku instalacji
            MOV  AH,Zamelduj       ;wywołaj funkcję PC-DOS,emitu-
            INT  UslugiDOS         ;jącą meldunki  o adresie [illegible] DX
            MOV  AH,UstawWekt      ;zapisz nowy wektor przerwania
            MOV  AL,1CH            ;nr 1CH,ustawiając go na wejś-
            MOV  DX,OFFSET Budzik  ;cie procedury obsługi budzika
            INT  UslugiDOS         ;korzystając  z  usług  PC-DOS
            MOV  DX,OFFSET Inicjuj;DX:=granica części rezydyjącej
            [illegible] Pozostaw   ;uczyń   program   rezydującym

MeldBlPar: [illegible] 7,'[illegible]ieobecny lub wadliwy parametr!',ODH,OAH,'$'
MeldInst:  DB   'Program zainstalowano [illegible] pamieci.',ODH,OAH,'$'
MeldNast:  DB   'Zmieniono nastawe budzika.',ODH,OAH,'$'

Kod_Prog   [illegible]

            END  Start

;******************* BUDZIK  wersja 1.01 *********************
;*********** Roland Waclawek,  Siemianowice Sl. 1987 **********

ZegarBIOS [illegible]  1AH         ;nr przerwania  obsługi  zegara BIOS
UslugiDOS [illegible]  21H         ;nr przerwania  wywołania  usług DOS
PowrotDOS EQU  20H         ;nr przerwania  zakończenia programu
Pozostaw  EQU  27H         ;nr fn.  czyniącej program rezdującym
LadujWekt [illegible]  35H         ;numer funkcji odczytującej  wektory
UstawWekt [illegible]  25H         ;numer funkcji zmieniającej  wektory
Zamelduj  [illegible]  9           ;nr funkcji PC-DOS emitującej napisy

Kod_prog  [illegible] PARA
          ASSUME CS:Kod_prog
          ORG  129
Parametry LABEL [illegible]        ;bufor parametrów wywołania programu
```

gdy chcemy całkowicie wyłączyć budzik albo wtedy, gdy obsługa budzika już trwa, ale wskutek jej przeciągnięcia (np. w wyniku obsługi innych przerwań) grozi ponowne jej wywołanie przed zakończeniem wywołania bieżącego. Dlatego na czas porównywania czasów ustawiany jest bit nr 0 tej zmiennej. Bit nr 1 zostanie trwale ustawiony po wykonaniu alarmu. Bit ten jest też ustawiany na czas wpisywania nowego czasu alarmu po wywołaniu programu BUDZIK.COM, co zapobiega przypadkowemu wywołaniu alarmu o niewłaściwej porze po wpisaniu godzin, ale jeszcze przed wpisaniem minut. W tym miejscu należałoby dodać, że alarm zostanie spowodowany nie tylko w razie zgodności obu czasów, ale także wtedy, gdy podany czas alarmu jest wcześniejszy od czasu bieżącego. Sygnał dźwiękowy zabrzmi wówczas niezwłocznie, sygnalizując w ten sposób prawdopodobny błąd.

Jeszcze jedna sprawa: jak odbywa się porównywanie czasów? Wywołując przerwanie 1AH otrzymujemy czas bieżący w postaci liczby impulsów zegarowych zarejestrowanych od godziny 0:00. W celu porównania z zaprogramowanym czasem alarmu musimy przetworzyć to na minuty i godziny. Niestety, na minutę nie przypada skończona liczba impulsów. Procesor 8086/88 nie posiada zaś rozkazów operujących na liczbach ułamkowych. Aby zwiększyć precyzję porównania, najpierw mnożymy odczytaną liczbę impulsów przez 8, a potem dzielimy przez liczbę impulsów przypadających na 1 minutę, także pomnożoną przez 8. Tak czy inaczej musimy zrezygnować z części ułamkowej dzielnika, ale względny błąd wynikający z jej odrzucenia jest w tym przypadku średnio rzecz biorąc 8 razy mniejszy (średni błąd zaokrąglenia jest odwrotnie proporcjonalny do wartości zaokrąglanej liczby). W wyniku dzielenia otrzymamy iloraz przedstawiający liczbę minut, które upłynęły od godziny 0:00. Teraz wystarczy podzielić go przez 60, aby nowy iloraz reprezentował liczbę godzin, a reszta z dzielenia — liczbę minut. Jeszcze jedno: ponieważ przy porównywaniu czasów badamy nie tylko ich zgodność, ale także ewentualną większość, liczba godzin musi być zapisana w bajcie bardziej znaczącym!

Gwoli ścisłości należałoby zaznaczyć, że PC-DOS dostarcza funkcji usługowych, podających bieżący czas wprost w godzinach, minutach i sekundach (funkcja nr 2CH przerwania nr 21H). Wywołania DOS są jednak znacznie bardziej czasochłonne, no i sprawa, powiedziałbym, pryncypialna (zarzekam się, że nie jestem tzw. czynnikiem!): PC-DOS nie jest systemem wieloprogramowym, a zatem jego funkcje usługowe nie muszą nadawać się do wywoływania z procedur obsługi przerwania. W przypadku funkcji 2CH nie miałoby to raczej większego znaczenia, ale nasze rozwiązanie jest przynajmniej formalnie poprawne.

Opisany program, co prawda w wersji bardziej rozbudowanej, był eksploatowany we współpracy z wieloma programami, w tym MS-WORD 3.0 i FRAMEWORK, które także „przeginają" wektor 1CH. Skutek był jak najbardziej pozytywny, przynajmniej z punktu widzenia TVP (zwiększona oglądalność Teleexpressu i Smurfów). Program został zapisany w konwencji przeznaczonej do uruchamiania w formacie .COM. Aby uzyskać działający program typu .COM, należy użyć kolejno asemblera (np. MASM), konsolidatora (np. LINK) oraz przetworzyć wyprodukowany przez konsolidator plik typu .EXE na program typu .COM za pomocą programu EXE2BIN.

```
            ORG   256
Start:      JMP   Inicjuj                  ; skok do części inicjacyjnej

Haslo:      DB    'R.WAC0288'              ; identyfikator części rezyd.
Minuty:     DB    0                        ; czas alarmu - pamięć minut
Godziny:    DB    0                        ; czas alarmu - pamięć godzin
Fl_Aktywn:  DB    [illegible]              ; semafor zabraniający obsługi
FL_Sygnal:  DB    0                        ; semafor generacji dźwięku
WektorOfs:  DW    [illegible]              ; adres starego wektora prze-
WektorSeg:  DW    [illegible]              ; rwania zegarowego nr 1CH
Licznik:    DB    18                       ; licznik ignorowanych cykli

Budzik:     TEST  [illegible] [illegible] FL_Sygnal, [illegible]  ; trwa emisja dźwięku?
            JNZ   Dzwiek                   ; tak-skok do obsługi dźwięku
            TEST  BYTE PTR Fl_Aktywn, [illegible]  ; wejście zabronione?
            JE    [illegible]              ; jeśli nie, to obsłuż budzik
E0:         JMP   Koniec                   ; tak - pomiń obsługę budzika

E1:         DEC   [illegible] PTR Licznik  ; odlicz kolejną 1/18 sekundy
            JNE   E0                       ; skok, gdy licznik niezerowy
            MOV   [illegible] [illegible] Fl_Aktywn, [illegible] ; zabroń powtórnego wejścia
            STI                        ; odblokuj przerwania maskowalne
            PUSH  AX                   ; przechowaj rejestry na stosie
            PUSH  CX
            PUSH  DX
            XOR   AH, AH               ; zeruj rejestr AH - funkcja nr 0
            INT   ZegarBIOS            ; odczytaj czas z BIOS do CX:DX
            MOV   AX, [illegible]      ; przenieś liczbę rejestrowanych
            MOV   DX, CX               ; impulsów z pary CX:DX do DX:AX
            ADD   AX, AX               ; pomnóż liczbę zawartą [illegible] parze
            ADC   DX, [illegible]      ; rejestrów DX:AX przez 8 drogą
            ADD   AX, AX               ; trzykrotnego dodawania jej do
            ADC   DX, DX               ; samej siebie
            ADD   AX, AX
            ADC   DX, [illegible]      ; DX:AX zawiera liczbę cykli x 8
            MOV   CX, 8739             ; dzielnik (8739.12=18.2065*60*8)
            DIV   CX                   ; AX=liczba minut od godziny 0:00
            MOV   CL, 60               ; podziel liczbę minut przez 60
            DIV   CL                   ; AH=minuty czasu akt., AL=godziny
            XCHG  AH, AL               ; minuty do rej. AL, godziny do AH
            CMP   AX, WORD PTR Minuty  ; porównaj czas akt. i alarmu
            JC    BezAlarmu            ; skok, jeśli czas alarmu>czas akt.
            MOV   BYTE PTR FL_Sygnal, 1   ; ustaw semafor gen. tonu
            MOV   BYTE PTR Licznik, 18*6  ; ładuj licznik cykli tonu
            CLI                        ; zabroń reakcji na przerwania IRQ
            MOV   BYTE [illegible] FL_Aktywn, 00000010B   ; wyłącz budzik
            JMP   Powrot               ; to już koniec obsługi budzika

Dzwiek:     PUSH  AX                   ; przechowaj na stosie rejestr AX
            DEC   [illegible] PTR Licznik   ; odlicz kolejny cykl tonu
            JS    StopTon              ; skok, jeżeli licznik już ujemny
            TEST  [illegible] [illegible] Licznik, 1  ; czy licznik jest parzysty?
            CLI                        ; zabroń reakcji na przerwania IRQ
            JNZ   WlaczTon             ; licznik nieparzysty-włącz dźwięk
            IN    AL, 61H              ; licznik parzysty - wyłącz dźwięk
            AND   AL, 11111100B        ; przez wyzerowanie bitów nr 0 i 1
            OUT   61H, AL              ; w rejestrze nr 61H
            JMP   KoniecTon            ; opuść procedurę obsługi dźwięku

WlaczTon:   MOV   AL, [illegible]      ; zaprogramuj dzielnik częstotliw.
            OUT   43H, AL              ; wpisując bajt ster. do portu 43H
            MOV   AL, [illegible]      ; zapisz dwubajtowy współczynnik
            OUT   42H, AL              ; podziału częstotliwości do re-
            OUT   42H, AL              ; jestru nr 42H (kolejno 2 bajty)
            IN    AL, 61H              ; rozpocznij generację dźwięku
            OR    AL, 11B              ; przez ustawienie bitów nr 0 i 1
            OUT   61H, AL              ; w rejestrze nr 61H
            JMP   KoniecTon            ; opuść procedurę obsługi dźwięku

StopTon:    MOV   [illegible] [illegible] FL_Sygnal, 0  ; zeruj semafor gen. tonu
            JMP   KoniecTon                ; opuść program generacji

BezAlarmu:  MOV   [illegible] [illegible] FL_Aktywn, 0  ; umożliw ponowne wejście
            MOV   [illegible] PTR Licznik, 18   ; inicjuj ponownie licznik
Powrot:     POP   DX                       ; odtwórz używane rejestry
            POP   CX
KoniecTon:  POP   AX
Koniec:     JMP   [illegible] [illegible] WektorOfs   ; skok do starej procedury
            ASSUME  DS:Kod_prog
Inicjuj:    MOV   AH, LadujWekt            ; wywołaj funkcję PC-DOS,
            MOV   AL, 1CH                  ; ładując wektor przerwa-
            INT   UslugiDOS                ; nia nr 1CH do rej. ES:BX
            MOV   [illegible] [illegible] WektorSeg, ES ; pamiętaj segment wektora
            MOV   [illegible] PTR WektorOfs, BX ; pamiętaj offset wektora
            MOV   SI, [illegible] Haslo    ; SI=offset identyfikatora
            MOV   DI, SI                   ; DI=offset identyfikatora
            MOV   CX, 4                    ; licz. porównywanych słów
            REPE  CMPSW              ; porównuj kod i identyfikator
            JE    Rezyduje           ; zgodność-program już rezyduje
            PUSH  DS                 ; skopiuj do rejestru [illegible] bie-
            POP   ES                 ; żącą zawartość rejestru DS
Rezyduje:   OR    [illegible] [illegible] ES:Fl_Aktywn, 10B; czasowo wyłącz budzik
            MOV   SI, OFFSET Parametry  ; adres parametrów do SI
            MOV   CH, 2              ; załaduj licznik dwukropków
            MOV   CL, 3              ; załaduj licznik cyfr godzin
            XOR   AH, AH             ; wyzeruj liczbę godzin w AH
LadujZnak:  LODS[illegible]          ; nast. bajt bl. param. do AL
            CMP   AL, [illegible]    ; czy koniec bloku parametrów?
            JE    KoniecPar          ; tak, bo znak <CR> kończy blok
            CMP   AL, ' '            ; czy kolejny znak jest spacją?
            JE    LadujZnak          ; jeśli tak, to zignoruj spację
            CMP   AL, ':'            ; czy ten znak to dwukropek?
            JE    Dwukropek          ; jeżeli tak, przejdź do minut
            CMP   AL, '9'            ; kod znaku większy niż kod 9?
            JA    BlednyPar          ; jeśli tak, podano zły parametr
            SUB   AL, '0'            ; AL:= kod znaku - kod cyfry 0
            JS    BlednyPar          ; różnica ujemna = zły parametr
            DEC   CL                 ; odlicz kolejną cyfrę w grupie
            JZ    BlednyPar          ; jeżeli liczba cyfr>2, to błąd
            ADD   AH, AH             ; pomnóż bieżącą wartość liczby
            ADD   AL, AH             ; przez 10, dodając do niej przy
            ADD   AH, AH             ; okazji wartość ostatnio wpro-
            ADD   AH, AH             ; wadzonej cyfry
            ADD   AH, AL
            JMP   LadujZnak          ; analizuj kolejny znak param.
```

```
Dwukropek:DEC   CH                    ;czy to pierwszy dwukropek?
          JNZ   PiszGodz              ;tak - pamiętaj godzinę alarmu
BlednyPar:MOV   DX,OFFSET MeldBlPar   ;DX:=adres melunku o błędzie
Bezlnstal:MOV   AH,Zamelduj           ;wywołaj funkcję PC-DOS, emitu-
          INT   UslugiDOS             ;jącą meldunki o adresie w DX
          INT   PowrotDOS             ;zwyczajny powrót do PC-DOS

PiszGodz: MOV   BYTE PTR ES:Godziny,AH  ;zapisz godzinę alarmu
          XOR   AH,AH                 ;wyzeruj liczbę minut w AH
          MOV   CL,3                  ;załaduj licznik cyfr minut
          JMP   LadujZnak             ;analizuj kolejny znak param.
KoniecPar:DEC   CH                    ;czy wystąpił już dwukropek?
          JNZ   BlednyPar             ;nie- nie podano minut - błąd!
          MOV   BYTE PTR ES:Minuty,AH   ;zapisz minutę alarmu
          MOV   BYTE PTR ES:Licznik,18  ;inicjuj licznik cykli
          AND   BYTE PTR ES:FL_Aktywn,11111101B
          MOV   AX,DS                 ;czy DS<>ES, co znaczy, że pro-
          CMP   AX,WORD PTR WektorSeg;gram jeszcze nie rezyduje
          MOV   DX,OFFSET MeldNast;DX:=adres meldunku o nastawie
          JE    BezInstal             ;skocz, jeżeli już rezyduje
          MOV   DX,OFFSET MeldInst;DX:=adres meldunku instalacji
          MOV   AH,Zamelduj           ;wywołaj funkcję PC-DOS, emitu-
          INT   UslugiDOS             ;jącą meldunki o adresie w DX
          MOV   AH,UstawWekt          ;zapisz nowy wektor przerwania
          MOV   AL,1CH                ;nr 1CH, ustawiając go na wejś-
          MOV   DX,OFFSET Budzik      ;cie procedury obsługi budzika
          INT   UslugiDOS             ;korzystając z usług PC-DOS
          MOV   DX,OFFSET Inicjuj;DX:=granica części rezydującej
          INT   Pozostaw              ;uczyń program rezydującym

MeldBlPar:DB    7,'Nieobecny lub wadliwy parametr!',0DH,0AH,'$'
MeldInst: DB    'Program zainstalowano w pamięci.',0DH,0AH,'$'
MeldNast: DB    'Zmieniono nastawę budzika.',0DH,0AH,'$'

Kod_Prog  ENDS
          END   Start
```

```
PROGRAM Loader_BUDZIKA;   {Roland Wacławek 88}

CONST kod_progr: ARRAY[1..28] OF STRING[32]=

      ('E9BC00522E574143303238380000000',
       '00000000122EF6060F01FF75582EF606',
       '0E01FF7403E992002EFE0E140175F62E',
       'C6060E0101FB50515232E4CD1A8BC28B',
       'D103C013D203C013D203C013D2B92322',
       'F7F1B13CF6F186E02E3B060C01724C2E',
       'C6060F01012EC60614016CFA2EC6060E',
       '0102EB4390502EFE0E140178252EF606',
       '140101FA7509E46124FCE661EB2B90B0',
       'B6E643B008E642E642E4610C03E661EB',
       '18902EC6060F0100EB0F902EC6060E01',
       '002EC6061401125A59582EFF2E1001B4',
       '35B01CCD218C061201891E1001BE0301',
       '8BFEB90400F3A774021E0726800E0E01',
       '02BE8100B502B10332E4AC3C0D74383C',
       '2074F73C3A74183C3977182C307814FE',
       'C9741002E402C402E402E402E0EBDBFE',
       'CD7509BA5C02B409CD21CD202688260D',
       '0132E4B103EBC3FECD75E82688260C01',
       '26C6061401122680260E01FD8CC03B06',
       '1201BAA20274CFBA7F02B409CD21B425',
       'B01CBA1501CD21BABF01CD27074E6965',
       '6F6265636E79206C7562207761646C69',
       '7779207061726 16D6574722 10D0A2450',
       '726F6772616D207A61696E7374616C6F',
       '77616E6F2077207061 6D69656369 2E0D',
       '0A245A6D69656E696F6E6F206E617374',
       '617765206275647A696B612E0D0A24');

VAR i,k,b,kod_bledu: Integer;
    bajt            : Byte;
    suma            : Real;
    progr           : FILE OF Byte;
BEGIN suma:= 0;
  Assign(progr, 'BUDZIK.COM'); ReWrite(progr);
  FOR i:=1 TO 28 DO
  FOR k:=1 TO Length(kod_progr[i]) DIV 2 DO
      BEGIN Val('$'+Copy(Kod_progr[i],k*2-1,2),
                  b, kod_bledu);      bajt:= b;
            Write(Progr,bajt); suma:= suma+bajt
      END;
  IF suma= 41414.0
    THEN BEGIN Close(progr); Writeln('Gotowe')
         END
    ELSE Writeln('Niepoprawny blok danych!')
END.
```


```
PROGRAM Loader_BUDZIKA;   {Roland Wacławek 88}
CONST kod_progr: ARRAY[1..28] OF STRING[32]=
      ('E9BC00522E574143303238380000000',
       '00000000122EF6060F01FF75582EF606',
       '0E01FF7403E992002EFE0E140175F62E',
       'C6060E0101FB50515232E4CD1A8BC28B',
       'D103C013D203C013D203C013D2B92322',
       'F7F1B13CF6F186E02E3B060C01724C2E',
       'C6060F01012EC60614016CFA2EC6060E',
       '0102EB4390502EFE0E140178252EF606',
       '140101FA7509E46124FCE661EB2B90B0',
       'B6E643B008E642E642E4610C03E661EB',
       '18902EC6060F0100EB0F902EC6060E01',
       '002EC6061401125A59582EFF2E1001B4',
       '35B01CCD218C061201891E1001BE0301',
       '8BFEB90400F3A774021E0726800E0E01',
       '02BE8100B502B10332E4AC3C0D74383C',
       '2074F73C3A74183C3977182C307814FE',
       'C9741002E402C402E402E402E0EBDBFE',
       'CD7509BA5C02B409CD21CD202688260D',
       '0132E4B103EBC3FECD75E82688260C01',
       '26C6061401122680260E01FD8CC03B06',
       '1201BAA20274CFBA7F02B409CD21B425',
       'B01CBA1501CD21BABF01CD27074E6965',
       '6F6265636E79206C7562207761646C69',
       '77792070617261 6D6574722 10D0A2450',
       '726F6772616D207A61696E7374616C6F',
       '77616E6F2077207061 6D69656369 2E0D',
       '0A245A6D69656E696F6E6F206E617374',
       '617765206275647A696B612E0D0A24');
VAR i,k,b,Kod_bledu: Integer;
    bajt            : Byte;
    suma            : Real;
    progr           : FILE OF Byte;
BEGIN suma:= 0;
  Assign(progr, 'BUDZIK.COM'); ReWrite(progr);
  FOR i:=1 TO 28 DO
    FOR k:=1 TO Length(kod_progr[i]) DIV 2 DO
      BEGIN Val('$'+Copy(Kod_progr[i],k*2-1,2),
                  b, kod_bledu);      bajt:= b;
            Write(Progr,bajt); suma:= suma+bajt
      END;
  IF suma= 41414.0
    THEN BEGIN Close(progr); Writeln('Gotowe')
         END
    ELSE Writeln('Niepoprawny blok danych!')
END.
```


Oto przykładowy ciąg zleceń, prowadzących do celu (zakładamy, że plik źródłowy nosi nazwę BUDZIK.ASM):

MASM budzik;
LINK budzik;
EXE2BIN budzik.exe budzik.com

Wysyłany przez konsolidator komunikat ostrzegawczy: *Warning: no STACK segment* jest w tym przypadku bez znaczenia. Jak zwykle, dla Czytelników niezbyt biegłych w asemblerze, ale zainteresowanych zabawą z programem zamieszczamy program ładujący w języku wysokiego poziomu. Tym razem jest to popularny TURBO-Pascal. Program ładujący napisano tak, że można go zrealizować zarówno w wersji 3.0, jak i 4.0 tego języka.

Po uruchomieniu program ładujący otwiera plik BUDZIK.COM i wpisuje do niego kolejne bajty kodu, zapisane w programie w formie szesnastkowych stałych. Każda stała składa się z dwóch cyfr i przed przekazaniem procedurze **Val**, która przetworzy ją na liczbę typu **Integer**, jest uzupełniana na początku symbolem '$', będącym w TURBO-Pascalu standardowym przedrostkiem liczb szesnastkowych. W razie gdyby przy „wpalcowywaniu" stałych popełniono błąd, program ładujący z wielkim prawdopodobieństwem wykryje ten fakt i zasygnalizuje go właściwym komunikatem.



JACEK RAUK

TURBO.LIB — BIBLIOTEKA PROCEDUR Turbo Pascala 3.0

Każdy użytkownik coraz bardziej popularnych u nas komputerów osobistych zgodnych ze standardem IBM PC wcześniej czy później odczuje potrzebę napisania własnego programu. Najlepszym, moim (i nie tylko moim) zdaniem, językiem dla początkującego programisty jest **Pascal**. O jego zaletach napisano już wiele, podobnie jak i zaletach najpopularniejszego kompilatora tego języka — **Turbo Pascala** firmy **Borland**. Nawet jednak tak udany program ma swoje „narowy": nie potrafi podnosić do potęgi, ma zwyczaj surowego karania pomyłek przy wprowadzaniu danych z klawiatury (podanie liczby rzeczywistej zamiast kropki dziesiętnej powoduje, że najlepiej nawet napisany program bezlitośnie odmawia współpracy), nie można kontrolować czasu systemowego (co bywa przydatne, gdy z jednego komputera korzysta kilka osób) itp. Firma Borland przewidziała wprawdzie szereg dodatkowych tricków, których zastosowanie daje w efekcie w pełni „idiotoodporny" program, zaś w polskiej prasie komputerowej opisano już wiele ułatwiających życie sztuczek, trudno jednak za każdym razem wertować opasłą dokumentację **TP** lub sięgać po roczniki czasopism. Nawet dokładne poznanie **Turbo Pascala** nie zwalnia zresztą od pracowitego wstukiwania podobnych sekwencji na początku każdego nowego programu. Jedynym sensownym wyjściem wydaje się stworzenie własnej biblioteki najpotrzebniejszych podprogramów.

Biblioteka **Turbo.Lib** powstała z procedur i funkcji, jakie znalazłem w czasie wertowania polskich czasopism komputerowych (**„Informik", „Mikroklan", „Komputer", „Bajtek", „IKS"**) wzbogaconych o kilka moich pomysłów. W nazwach i w komunikatach o błędach zdecydowałem się na język angielski. W ten sposób uniknąłem dziwolągów powstających przy pisaniu polskich słów bez polskich znaków diakrytycznych z osławionym komunikatem — obelgą „Blad!" na czele. Treść procedur i funkcji przedstawia dołączony listing, oto zaś krótki ich opis (w przypadku cudzych pomysłów podałem w kwadratowych nawiasach numery czasopism i nazwisko autora artykułu, w którym zawarty jest dokładny opis ich działania):

Opis Biblioteki Turbo.Lib

Typy:

1. **Name = string [41]**;
— zmienne tego typu to głównie nazwy plików (podprogramy **FileExist**, **BackUp**, **Dir**, **ShowDir**). By nie tworzyć nowych typów wykorzystałem go też do wyprowadzenia czasu i daty systemowej (funkcje **Time**, **Date**, **Day**).
2. **Files = array [1..512] of Name**;
— tablica nazw plików umożliwiająca zapisanie 512 nazw (wykorzystywana w procedurach **Dir** i **ShowDir**).
3. **FilesPtr = ^Files**
— typ wskazujący na zmienną typu **Files**.

Procedury i funkcje:

1. Procedura **ErrorTL**.

Procedura ta służy wyłącznie do obsługi błędów, jakie powstać mogą na skutek złego użycia innych podprogramów biblioteki (np. podniesienie zera do ujemnej potęgi). Jej działanie polega na wyprowadzeniu komunikatu o błędzie:

„TURBO.LIB ERROR No. *Nr_błędu*
Program aborted. Sorry..."

i przerwaniu działania programu.

Podana w komunikacie wartość *Nr_błędu* oznacza odpowiednio:

1 — podanie zbyt dużego argumentu funkcji **Factorial** (> 33);
2 — podanie ujemnego argumentu funkcji **Factorial**;

3 — zero do ujemnej potęgi w funkcji **Power**;
4 — podanie w funkcji **Power** ujemnej podstawy i niecałkowitego wykładnika potęgi.

2. Procedura **Gong**.
Wywołanie: **Gong** (*n*);
gdzie *n* — wartość lub zmienna typu **integer**.

Wywołanie procedury powoduje *n*-krotny sygnał akustyczny (niedodatnie wartości *n* traktowane są jak $n = 1$). Procedura wykorzystana być może jako „budzik" informujący o zakończeniu obliczeń lub wołający użytkownika programu w chwilach, gdy konieczna jest jego ingerencja.

3 i **4**. Procedury **RealRead** i **RealReadln**.
Wywołanie: **RealRead** (*x*);
RealReadln (*x*);
gdzie *x* — zmienna typu **real**.

Procedury **3** i **4** służą do „bezpiecznego" wczytywania z klawiatury wartości rzeczywistych odpowiednio bez i z przeniesieniem kursora do nowego wiersza. Podanie nieprawidłowej (niemożliwej do zaakceptowania przez program) wartości (np. 2,0003, 2.4.5 itp.) nie przerywa działania programu, lecz powoduje zmazanie błędnego zapisu i oczekiwanie na nowy. Towarzyszy temu zwracający uwagę na wystąpienie błędu sygnał akustyczny. W odróżnieniu od procedury **Read** i **Readln** wczytana może być tylko 1 zmienna.

5 i **6**. Procedury **IntRead** i **IntReadln**.
Wywołanie: **Int Read** (*n*);
IntReadln (*n*);
gdzie *n* — zmienna typu **integer**.

Procedury **5** i **6** służą do „bezpiecznego" wczytywania wartości całkowitych. Działanie podobne do **RealRead** i **RealReadln**.

7 i **8**. Procedury **BoolRead** i **BoolReadln**.
Wywołanie: **BoolRead** (*x*);
BoolReadln (*x*);
gdzie *x* — zmienna typu **Boolean**.

Procedury **7** i **8** służą do wczytywania wartości logicznych (np. decyzji użytkownika programu) odpowiednio bez — i z przeniesieniem kursora do nowego wiersza. Wciśnięcie klawiszy **[T]** (jak „**true**", „**tak**") i **[Y]** (jak „**yes**") nadaje zmiennej *x* wartość **true**, zaś **[F]** (jak „**false**") i **[N]** (jak „**no**", „**nie**") wartość **false**. Pozostałe klawisze są ignorowane.

9. Funkcja **Factorial** (typu **real**).
Wywołanie: **Factorial** (*n*);
gdzie *n* — zmienna lub wartość typu **integer**.

Rezultatem funkcji **Factorial** jest dana typu **real** równa *n!* (**n silnia**). Ze względu na szybko rosnącą ze wzrostem *n* wartość funkcji *n* nie może być większe od 33 i mniejsze od zera. Wykroczenie poza ten zakres spowoduje przerwanie programu i komunikat o błędzie nr 1 ($n > 33$) lub 2 ($n < 0$).

10. Funkcja **Power** (typu **real**).
Wywołanie: **Power** (*a,b*);
gdzie *a* i *b* — zmienne lub wartości typu **real**.

Rezultatem funkcji **Power** jest dana typu **real** będąca wynikiem podniesienia *a* do potęgi *b*. Dla dodatnich wartości podstawy *a* wykładnik *b* przyjmować może dowolne wartości rzeczywiste, dla $a = 0$ wartości rzeczywiste nieujemne zaś dla $a < 0$ wartości całkowite (wartości te mogą być typu **real** nie mogą jednak zawierać części dziesiętnej). Próba podniesienia zera do ujemnej potęgi powoduje przerwanie programu i komunikat o błędzie nr 3 (procedura **Error**), zaś podniesienie wartości ujemnej do potęgi niecałkowitej przerwanie programu i komunikat o błędzie nr 4.

11. Funkcja **FileExist** (typu **Boolean**).
Wywołanie: **FileExist** (*Filename*);
gdzie *Filename* — zmienna typu **Name** — nazwa pliku.

Rezultatem funkcji **FileExist** jest wartość logiczna **true**, gdy plik o podanej nazwie *Filename* istnieje i **false**, gdy nie istnieje. Nazwa *Filename* zawierać może ścieżkę dostępu (np. *Filename* = **'C:\TPASCAL\JACEK\TURBO.LIB'**). Brak ścieżki dostępu powoduje szukanie pliku w aktualnym katalogu.

12. Procedura **BackUp**.
Wywołanie: **BackUp** (*Filename*);
gdzie *Filename* — zmienna typu **Name** — nazwa pliku.

Wywołanie procedury **BackUp** powoduje zmianę rozszerzenia nazwy pliku *Filename* na **.BAK** o ile plik o nazwie *Filename* już istnieje. Wywołanie procedury **BackUp** przed każdym otwarciem zbioru do zapisu zabezpiecza przed przypadkową utratą starej zawartości zbioru.

14. Procedura **PrtScr**.
Wywołanie: **PrtScr**;
Wywołanie tej procedury jest równoważne jednoczesnemu wciśnięciu klawiszy [SHIFT] i [PrtSc] — powoduje wydruk zawartości ekranu na drukarce. **[R. Wacławek; INFORMIK nr 4/1987]**.

15. Funkcja **PrinterOK** (typu **Boolean**).
Wywołanie: **PrinterOk**;
Funkcja **PrinterOK** jest bezparametrowa. Rezultatem jej jest wartość logiczna **true**, gdy dołączona do komputera drukarka jest gotowa do pracy lub wartość **false**, gdy drukarka jest uszkodzona, brakuje papieru itp. **[R. Wacławek; KOMPUTER nr 3/1988]**.

16. Funkcja **DDriveOK** (typu **Boolean**).
Wywołanie: **DDriveOK** (*Drive*);
gdzie *Drive* — zmienna lub wartość typu **char** — oznaczenie stacji dysków elastycznych.

Rezultatem funkcji **DDriveOK** jest wartość logiczna **true**, jeżeli dyskietka w napędzie *Drive* jest gotowa do odczytu i zapisu lub wartość **false**, jeżeli stacja jest otwarta, dyskietka nie jest sformatowana itp. **[R. Wacławek; KOMPUTER nr 3/1988]**.

17. Funkcja **Time** (typu **Name**).
Wywołanie: **Time**;





Funkcja **Time** jest bezparametrowa. Jej rezultatem jest dana typu **Name**, przedstawiająca czas systemowy w postaci łańcucha:

hh:mm:ss

gdzie *hh* — godzina, *mm* — minuty, *ss* — sekundy (np. **13:45:00** oznacza godzinę 13 minut 45). **[R. Wacławek; INFORMIK nr 1/1987]**.

18. Funkcja **Date** (typu **Name**).
Wywołanie: **Date**;
Rezultatem funkcji **Date** jest dana typu **Name** przedstawiająca datę systemową w postaci łańcucha:

rrrr.mm.dd

gdzie *rrrr* — rok, *mm* — miesiąc, *dd* — dzień miesiąca (np. łańcuch **1988.07.12** oznacza 12 lipca 1988 roku).

19. Funkcja **Day** (typu **Name**).
Wywołanie: **Day**;
Rezultatem funkcji **Day** jest dana typu **Name** przedstawiająca dzień tygodnia według daty systemowej w postaci jego angielskiej nazwy.

20. Procedura **Dir**.
Wywołanie: **Dir** (*DirMask*, *pointer*);
gdzie
DirMask — zmienna lub wartość typu **Name**, oznaczająca szablon nazw plików, których wykaz chcemy uzyskać (analogicznie jak w DOS-ie stosować można znaki * i ?, ścieżkę dostępu itp.).
pointer — zmienna typu **FilesPtr**, która po wykonaniu procedury wskazywać będzie na tablicę nazw plików.

Wywołanie procedury powoduje utworzenie zmiennej tablicowej typu **Files**, wskazywanej przez zmienną *pointer* i umieszczenie w niej nazw plików zgodnych z podanym szablonem *DirMask*. Nazwy te nie są wyświetlane na ekranie, mogą jednak służyć np. do analizy aktualnego katalogu, wyszukiwania nazw plików o podanym rozszerzeniu itp. Do wyświetlania katalogu na ekranie służy następna procedura: **ShowDir**. Tablica przechowująca nazwy plików zajmuje niestety dosyć dużo pamięci, którą jednak można odzyskać, wykonując po wykorzystaniu potrzebnych informacji predefiniowaną w Turbo Pascalu procedurę **Dispose(***pointer***)**. **[R. Wacławek, INFORMIK nr 1/1988]**.

21. Procedura **ShowDir**.
Wywołanie: **ShowDir** (*DirMask*);
gdzie
DirMask — zmienna lub wartość typu **Name**, oznaczająca szablon nazw plików, których wykaz chcemy uzyskać na ekranie (analogicznie jak w procedurze **Dir**). Wywołanie procedury **ShowDir** powoduje wyświetlenie na ekranie spisu nazw plików zgodnych z podanym schematem *DirMask*. Nazwy plików poprzedzone są kolejnym numerem i wyświetlane po cztery w linii. Jeżeli liczba plików przekracza pojemność ekranu, wtedy wyświetlanie kolejnych nazw zostaje wstrzymane, a na dole ekranu ukazuje się pytanie:

More?[Y]es/[N]o

Udzielenie odpowiedzi twierdzącej (wciśnięcie klawisza **[Y]**) powoduje kontynuację wyprowadzania nazw plików na ekran. Odpowiedź przecząca (wciśnięcie klawisza **[N]**) powoduje wyjście z procedury. Po wykonaniu procedury **ShowDir** pamięć wykorzystywana przez wywoływaną z niej procedurę **Dir** jest automatycznie odzyskiwana.

* * *

Gotową bibliotekę wystarczy dołączyć na początku programu (najlepiej zaraz po **nagłówku**) dyrektywą:

{$I Turbo.Lib}

i spokojnie korzystać już można ze wszystkich jej podprogramów. Dołączenie biblioteki zwiększa rozmiar gotowego, skompilowanego programu o **3705 bajtów**. To chyba niezbyt wysoka cena za usługi, jakie może oddać. Zachęcam do rozszerzenia biblioteki o własne podprogramy oraz do tworzenia odrębnych, nastawionych na rozwiązywanie konkretnych problemów zawodowych, bibliotek. To naprawdę oszczędza czas.

Przypis redakcji:
Brak procedury o numerze 13 NIE jest spowodowany błędem drukarskim, ale względami formalnymi narzuconymi przez Autora, co redakcja w pełni zaakceptowała.

```
{                                                    }
{               TURBO.LIB  1.0                       }
{     biblioteka procedur Turbo Pascala              }
{                         J.Rauk   1988              }
{                                                    }
Type
 Name     = string [41];
 Files    = array [1..512] of Name;
 FilesPtr = ^Files;

procedure        ErrorTL ( n : integer );
{ obsluga bledow biblioteki               }
{ Numery bledow:                          }
{ 1 - argument funkcji Factorial          }
{     zbyt duzy;                          }
{ 2 - ujemny argument funcji              }
{     Factorial;                          }
{ 3 - zero do ujemnej potegi              }
{     w funcji Power;                     }
{ 4 - w funcji Power ujemna               }
{     podstawa i niecalkowity             }
{     wykladnik.                          }
 begin
  Writeln;
  Writeln
       ( ' TURBO.LIB ERROR No.',n:2 );
  Write ( #7 );
  Writeln
      ( ' Program aborted. Sorry...');
  Halt
 end;                           { ErrorTL }

procedure         Gong ( n : integer );
{ procedura  powtarza  n razy             }
{ sygnal akustyczny                       }
 var
  i : byte;
 begin
  If n < 1 then n := 1;
  For  i := 1 to n do
   begin
    Write ( #7 );
    Delay ( 300 )
   end
 end;                              { Gong }

procedure     RealRead ( var x : real );
{ procedura wczytuje wartosc              }
{ rzeczywista pozostawiajac kursor        }
{ w tej samej linii                       }
 var
  OK   : Boolean;
  z,y : byte;
 begin
   z := WhereX;
   y := WhereY;
   Repeat
 {$I-}
    Read ( x );
 {$I+}
    OK := ( IOResult = 0 );
    If not OK then
     begin
      Gong ( 1 );
      GotoXY ( z,y );
      ClrEol
     end
   Until OK
 end;                          { RealRead }

procedure   RealReadln ( var x : real );
{ procedura wczytuje wartosc              }
{ rzeczywista i przenosi kursor           }
{ do nastepnej linii                      }
 begin
  RealRead ( x );
  Writeln
 end;                        { RealReadln }

procedure   IntRead ( var x : integer );
{ procedura wczytuje wartosc              }
{ calkowita pozostawiajac kursor          }
{ w tej samej linii                       }
 var
  OK  : Boolean;
  z,y : byte;
 begin
  z := WhereX;
  y := WhereY;
  Repeat
 {$I-}
  Read ( x );
 {$I+}
  OK := ( IOResult = 0 );
  If not OK then
   begin
    Gong ( 1 );
    GotoXY ( z,y );
    ClrEol
   end
  Until OK
 end;                           { IntRead }

procedure IntReadln ( var x : integer );
{ procedura wczytuje wartosc calkowita }
{ i przenosi kursor do nastepnej linii }
 begin
  IntRead ( x );
  Writeln
 end;                         { IntReadln }

procedure BoolRead ( var  x  : Boolean);
{ procedura wczytuje wartosc logiczna  }
{ pozostawiajac kursor                 }
{ w tej samej linii                    }
{     znaczenie klawiszy :             }
{      'T', 'Y' - true                 }
{      'F', 'N' - false                }
 var
  znak : char;
 begin
  Repeat
   Read ( Kbd,znak );
   Case UpCase ( znak ) of
    'T','Y'  : x := true;
    'F','N'  : x := false;
   else
    Gong ( 1 )
   end;
  Until
   UpCase ( znak ) in ['T','Y','F','N'];
  Write ( znak )
 end;                          { BoolRead }
procedure BoolReadln ( var x : Boolean);
{ procedura wczytuje wartosc logiczna  }
{ i przenosi kursor do nastepnej linii }
{     znaczenie klawiszy :             }
{      'T', 'Y' - true                 }
{      'F', 'N' - false                }
 var
  znak : char;
 begin
  Repeat
   Read ( Kbd,znak );
   Case UpCase ( znak ) of
    'T','Y'  : x := true;
    'F','N'  : x := false;
   else
    Gong ( 1 );
   end;
  Until
   UpCase ( znak ) in ['T','Y','F','N'];
  Writeln ( znak )
 end;                        { BoolReadln }

function Factorial ( n: integer ): real;
{ funkcja liczy wartosc n!             }
 begin
  If n>33 then ErrorTL ( 1 )
  else
   If  n<0 then ErrorTL ( 2 )
   else
    If n>0 then
     factorial := n * factorial ( n-1 )
    else
     factorial := 1
 end;                         { Factorial }
```

```
function
     Power ( base,index : real ) : real;
{ funkcja oblicza wartosc base^index    }
 begin
  If index = 0 then
   Power := 1
  else
   If base = 0 then
    If index < 0 then ErrorTL ( 3 )
    else
    Power := 0
   else
    If base > 0 then
     Power := Exp ( index * Ln( base ))
    else
     begin
      If  index = Trunc ( index ) then
       begin
        If index >0 then
         Power :=
          Power ( base,index-1 )*base;
        If index <0 then
        Power :=
         1/Power ( base, Abs ( index ))
       end;
      If index <> Trunc ( index ) then
       ErrorTL ( 4 )
     end
 end;                          { Power }


function
 FileExist ( FileName: Name ): Boolean ;
{ funkcja sprawdzajaca istnienie pliku }
{ o nazwie FileName                    }
 var
  plik : file;
 begin
  Assign ( plik,FileName );
{$I-}
  Reset ( plik );
{$I+}
  FileExist := ( IOResult=0 );
  If IOResult=0 then Close ( plik )
 end;                      { FileExist }

procedure    BackUp ( FileName : Name );
{ procedura zmienia rozszerzenie       }
{ nazwy pliku na BAK.                  }
 var
  plik : file;
  FN   : Name;
  x    : integer;
 begin
  If FileExist ( FileName ) then
   begin
    x := Pos ('.',FileName);
    If x = 0 then
     FN := Concat ( FileName,'.BAK' )
    else
     begin
      FN := Copy ( FileName,1,x );
      FN := Concat ( FN,'BAK')
     end;


    If FileExist ( FN ) then
     begin
      Assign ( plik,FN );
      Erase  ( plik )
     end;
    Assign ( plik,FileName );
    Rename ( plik, FN);
   end
 end;                         { BackUp }

procedure                       PrtScr;
{ procedura wywoluje przerwanie nr 5   }
{ - odpowiednik wcisniesia klawiszy    }
{          SHIFT + PrtSc               }
{ R.Waclawek           INFORMIK 4/1987 }
 begin
     InLine ( $CD/5 )
 end;                         { PrtScr }
```

```
function             PrinterOK : Boolean;
{ funkcja testuje sprawnosc drukarki    }
{ na podst:                             }
{ R.Waclawek         KOMPUTER 3/1988    }
 var
  Rejestry : record
             AL,AH,BL,
             BH,CL,CH        : byte;
             DX,BP,SI,
             DI,DS,ES,flagi  : integer
            end;
  OK       : Boolean;
 begin
  Rejestry.AH := 2;
  Rejestry.DX := 0;
  Intr ( $17,Rejestry );
  With Rejestry do
   begin
    If AH and $08 <>0 then
     PrinterOK := false
   else
    If AH and $01 <>0 then
     PrinterOK := False
    else
     If AH and $20 <>0 then
      PrinterOK := False
     else
      If AH and $10 = 0 then
       PrinterOK := false
      else
       PrinterOK := true
  end
end;                        { PrinterOK }


function
     DDriveOK ( Drive : char ) : Boolean;
{  testuje sprawnosc napedu dyskow      }
{ na podst:                             }
{ R.Waclawek            KOMPUTER 3/1988 }
 var
  Rejestry : record
             AL,AH,BL,
             BH,CL,CH,DL,DH : byte;
             BP,SI,DI,
             DS,ES,flagi    : integer
            end;
 begin
  Drive := UpCase ( Drive );
  With Rejestry do
   begin
AH := 4; AL := 1;
CH := 0; CL := 1;
DH := 0;
Dl := Ord ( Drive ) - 65;
Intr ( $13,Rejestry );
If AH > 0 then
  begin


     AH := 4; AL := 1;
     CH := 0; CL := 1;
     DH := 0;
     Dl := Ord ( Drive ) - 65;
     Intr ( $13,Rejestry );
    end;
   If AH > 0 then
    DDriveOK := false
   else
    DDriveOK := true;
  end
 end;                        { DDriveOK }

function                    Time : Name;
{ funkcja podaje czas systemowy         }
{ R.Waclawek  INFORMIK 1/1988           }
 var
  Rejestry : record
             AL,AH,BL,
             BH,CL,CH,DL,DH : byte;
             BP,SI,DI,
             DS,ES,flagi    : integer
            end;
```

```
  h,m,s     : string [2];
 begin
  Rejestry.AH := $2c;
  MsDos ( Rejestry );
  With Rejestry do
   begin
    Str ( CH,h );
    Str ( CL,m );
    Str ( DH,s )
   end;
  If Length ( ■ ) = 1 then ■ := '0'+m;
  If Length ( ■ ) = 1 then s := '0'+s;
  Time := h+':'+m+':'+s
 end;                                    { Time }

function                          Date : Name;
{ funkcja podaje date systemowa            }
 var
  Rejestry : record
              AL,AH,BL,BH     : byte;
              CX              : integer;
              DL,DH           : byte;
              BP,SI,
              DI,DS,ES,flagi  : integer
             end;
  mm,dd     : string[2];
  y         : string[4];
 begin
  Rejestry.AH := $2a;
  MsDos ( Rejestry );
  With Rejestry do
   begin
    Str ( DL,dd );
    Str ( DH,mm );
    Str ( CX, y );
   end;
  If Length ( ■■ ) = 1 then
   ■■ := '0'+mm;
  If Length ( dd ) = 1 then
   dd := '0'+dd;
  Date := y+'.'+mm+'.'+dd
 end;                                    { Date }

function                           Day : Name;
{ funkcja podaje dzien tygodnia            }
{ wg. daty systemowej                      }
 var
  Rejestry : record
              AL,AH,BL,BH     : byte;
              CX              : integer;
              DL,DH           : byte;
              BP,SI,
              DI,DS,ES,flagi  : integer
             end;
 begin
  Rejestry.AH := $2a;
  MsDos ( Rejestry );
  Case Rejestry.AL of
   0 : Day := 'Sunday    ';
   1 : Day := 'Monday    ';
   2 : Day := 'Tuesday   ';
   3 : Day := 'Wednesday';
   4 : Day := 'Thursday  ';
   5 : Day := 'Friday    ';
   ■ : Day := 'Saturday  ';
  end
 end;                                     { Day }


procedure
  Dir ( DirMask: Name;var D: FilesPtr );
{ Procedura znajduje nazwy plikow          }
{ zgodnych z szablonem DirMask             }
{ i umieszcza je w tablicy                 }
{ wskazywanej przez  zmienna FilesPtr      }
{ na podst .                               }
{ R.Waclawek          INFORMIK 1/1988      }
 var
  LastFile,i  : integer;
  DTA         : array [0..42] of char;
  Rejestry    : record
                 AL,AH          : byte;
                 BX,CX,DX,BP,
                 SI,DI,DS,ES,
                 flagi          : integer
                end;

 procedure NextFile;
  var
   No : integer;
  begin
   LastFile := LastFile + 1;
   No := 30;
   While DTA [ No ] <> #0 do
    begin
     D^[LastFile]:=D^[LastFile]+DTA[No];
     No := No + 1
    end
  end;    { NextFile }
{        poczatek procedury Dir
 begin
  New ( D );
  For i := 1 to 512 do
   D^[i] := '';
  DirMask  := DirMask + #0;
  LastFile := 0;
  With Rejestry do
   begin
    DS := Seg ( DTA );
    DX := Ofs ( DTA );
    CX := 0;
    AH := $1A;
    MsDos ( Rejestry );
    DS := Seg ( DirMask [1] );
    DX := Ofs ( DirMask [1] );
    AH := $4E;
    MsDos ( Rejestry );
    If AL = 0 then
     Repeat
      NextFile;
      AH := $4F;
      MsDos ( Rejestry )
     Until
      ( AL <> 0 ) or ( LastFile = 512 )
   end
 end;                                    { Dir }

procedure    ShowDir ( DirMask : Name );
{ Procedura wyswietla na ekranie liste     }
{ plikow o nazwie zgodnej z szablonem      }
{ DirMask                                  }
 var
  i  : integer;
  OK : Boolean;
  ■  : FilesPtr;
 begin
  Writeln;
  Dir ( DirMask,a );
  i := 1;
  Writeln ( 'Dir Mask : ',DirMask );
  Writeln;
  If a^ [1] = '' then
   Writeln ( ' No Files ! ');
  While a^ [i] <> '' do
   begin
    If Pos ( '.',a^ [i] ) = 0 then
     a^ [i] := a^ [i] + '.    ';
    Write ( i:5,a^ [i]:15 );
    If i mod 80 = 0 then
     begin
      Writeln;
      Write (' More ? [Y]es/[N]o ');
      Boolreadln ( OK );
      If not OK then
       begin
        Dispose ( ■ );
        Exit
       end
      else Writeln;
     end;
    i := i + 1
   end;
  Dispose ( a );
  a := nil
 end;                                  { ShowDir }
```



MIKROKOMPUTER JUNIOR — pamięć dyskowa

WŁADYSŁAW STRUGAŁA

Każdego użytkownika mikrokomputera, jeśli tylko miał możliwość porównania komfortu pracy komputera wyposażonego w stację dysków i komputera z magnetofonem jako zewnętrzną pamięcią masową, nie trzeba przekonywać o zaletach pierwszego rozwiązania. Mikrokomputer szkolny Elwro 800-2 Junior ma możliwość pracy zarówno z magnetofonem, jak i ze stacją dysków elastycznych 5.25" — dwa napędy (ang. *Floppy Disk Drive*, w skrócie FDD) po 720 KB!

Junior produkowany jest w dwóch wersjach: nauczycielskiej i uczniowskiej. Zarówno pierwsza, jak i druga wersja mają możliwość współpracy ze stacją dysków. Standardowo jednak (ze względu na znaczny koszt stacji) tylko komputer nauczycielski może bezpośrednio z nią współpracować. Komputery uczniowskie natomiast, mogą korzystać ze stacji poprzez sieć komputerową JUNET.

Struktura zapisu informacji na dyskach

Podstawowe parametry pamięci dyskowej mikrokomputera Elwro 800 Junior:
1. Dwie strony dysku: 0 i 1.
2. Każda strona ma 80 ścieżek.
3. Każda ścieżka posiada 90 sektorów.
4. Każdy sektor ma długość 512 bajtów.
5. Tzw. rekord jest umowną jednostką długości zbioru. Jeden rekord to 128 bajtów. Jeden sektor zawiera 4 rekordy.

Sektor jest najmniejszą porcją informacji, którą fizycznie można odczytać lub zapisać na dysku. Na jednej ścieżce mieści się więc dokładnie 9·512 = 4608 bajtów, czyli 4,5 KB informacji. Na jednej stronie mamy wobec tego 80·9 = 720 sektorów, czyli 80·4,5 = 360 KB, a na całym dysku 2·720 = 1440 sektorów, czyli 2·360 = 720 KB. Jest to pojemność maksymalna dysku. W rzeczywistości do dyspozycji użytkownika pozostaje mniej ze względu na wykorzystanie niektórych obszarów dysku na informacje systemowe (i nie tylko).

Dyskowy system operacyjny Juniora wymienia informacje ze stacją w porcjach zwanych blokami. Jeden blok to 2 KB. Blok zajmuje na dysku fizycznie 4 sektory. Na pojedynczej ścieżce mieści się wobec tego $2^1/_4$ bloku. Jeden blok może zajmować miejsce w całości na jednej ścieżce, jak również może być „rozrzucony" na dwóch ścieżkach (i na dwóch stronach!) dysku. Odpowiednich przeliczeń, gdzie fizycznie na dysku znajduje się określony blok, dokonuje w sposób niewidoczny dla użytkownika — system operacyjny w czasie zapisu lub odczytu informacji.

Oczywiście użytkownik komputera nie musi pamiętać, w których blokach zapisany jest jego program czy dane. To także załatwia za niego system operacyjny. Do wczytania lub zapisu programu lub danych wystarcza nazwa (pliku, zbioru).

Dysk podzielony jest na kilka zarezerwowanych obszarów:
1. Na ścieżce 0 (numeracja ścieżek od 0 do 79), na stronach 0 i 1 zapisywany jest system operacyjny CP/J.
2. Na ścieżce 1 na stronie 0 i w sektorach od 1 do 7 (numeracja sektorów na ścieżce od 1 do 9) na ścieżce 1 na stronie 1 (16 sektorów czyli 4 bloki) zapisany jest tzw. katalog dysku (ang. *directory*).
3. Od sektora 8 na ścieżce 1 na stronie 1 rozpoczyna się obszar przeznaczony na dane użytkownika.

Każdy zbiór użytkownika opisany jest w katalogu dysku w tzw. tablicach FCB (ang. *File Control Block*). Struktura FCB opisana jest dokładnie w instrukcji komputera. Pojedynczy zbiór użytkownika może być opisany przez jedną lub więcej tablic FCB (zależy to od wielkości danego zbioru). Pojedyncza tablica FCB składa się z 32 bajtów. W skrócie jej struktura jest następująca:
1. Bajt nr 0 — numer użytkownika lub E5h, jeśli zbiór jest skasowany. Umożliwia identyfikację zbiorów poszczególnych użytkowników sieci JUNET.
2. Bajty 1..11 — nazwa zbioru (8 bajtów) z rozszerzeniem (3 bajty) (razem 11 bajtów).
3. Bajt 12 — numer tablicy FCB opisującej dany zbiór. Pierwsza tablica FCB zbioru zawiera tutaj nr 0. Jeśli zbiór jest dłuższy niż 16 KB, wówczas opisany jest także w kolejnych tablicach FCB. Bajt 12 w następnych tablicach zawiera wówczas kolejne liczby 1, 2 itd. Zbiór o długości 16..32 KB opisany jest w dwóch tablicach FCB, 32..48 KB w trzech. Dłuższe zbiory raczej nie występują ze względu na ograniczenie pamięci operacyjnej komputera Junior.
4. Bajty 13..14 są zarezerwowane.
5. Bajt 15 podaje długość zbioru (lub fragmentu zbioru) opisanego daną tablicą FCB w rekordach.
6. Bajty 16..31 zawierają numery bloków, w których zapisany jest zbiór (lub fragment zbioru) opisany daną tablicą FCB. Każdy numer bloku zajmuje dwa bajty w tablicy. Jedna tablica FCB może więc opisywać zbiór (lub fragment zbioru) o wielkości maksimum 8 bloków, czyli 16 KB.

Katalog dysku zajmuje 8 KB (4 bloki — numery od 0 do 3). Może więc pomieścić 256 tablic FCB. Jeżeli każdy zbiór opisany byłby jedną tablicą FCB i miałby długość maksymalnie 2 KB, wówczas wykorzystując tylko 256·2 = 512 KB nominalnej pojemności dysku można go całkowicie wypełnić. Najkrótszy nawet zbiór (1 bajt) zajmuje na dysku 1 blok (2 KB) i jedną tablicę FCB. Należy mieć to na uwadze i — w miarę możliwości oczywiście — nie „zaśmiecać" dysku bardzo krótkimi zbiorami.

Mikrokomputer Junior ma dwa tryby pracy: ZX Spectrum i CP/J. W trybie ZX Spectrum ze stacji dysków może korzystać bezpośrednio jedynie komputer, który wyposażony jest w płytkę tzw. kontrolera (sterownika) stacji dysków (ang. (*Floppy Disc Controller*, w skrócie FDC), i do którego jest ona podłączona (zwykle nauczycielski). Odczyt i zapis zbiorów dokonuje się wówczas przy pomocy poleceń o składni podobnej jak przy współpracy z magnetofonem tzn. LOAD, MERGE, SAVE. Po słowach tych należy jedynie umieścić gwiazdkę *, a nazwa zbioru nie może mieć więcej jak 8 znaków.

Dodając po nazwie CODE informujemy komputer, że chodzi o zbiór bajtów, DATA — zmienne i tablice (jak w przypad-

ku magnetofonu). Na podstawie tych informacji system operacyjny w trybie ZX Spectrum dokonuje zapisu lub odczytu zbioru określonego typu (z odpowiednim rozszerzeniem nazwy). W tablicach FCB zbiory typu ZX Spectrum posiadają rozszerzenia nazwy PRG, COD, ARR, STR.

Pierwsze bloki zbiorów typu ZX Spectrum zawierają zawsze na początku znane użytkownikom komputera ZX Spectrum nagłówki o identycznej jak w przypadku magnetofonu długości (17 bajtów) i strukturze.

Jeżeli zapisujemy na dysk w trybie ZX Spectrum zbiór np. bajtów o długości dokładnie 2 KB, to zostanie on fizycznie zapisany w dwóch blokach (bo 17 bajtów nagłówka + 2 KB danych to już więcej niż 2 KB). W pierwszym bloku zostanie zapisany najpierw nagłówek (17 bajtów), a następnie 2048-17-2031 bajtów danych. W drugim bloku umieszczone zostanie pozostałe 17 bajtów danych.

Normalnie użytkownik komputera nie ma możliwości „podglądania" informacji zapisanych np. na ścieżkach systemowych lub w katalogu dysku. Może jedynie zapisywać, czytać lub kasować zbiory o określonej nazwie, lub też czytać nazwy i niektóre parametry zbiorów zapisane w katalogu dysku (polecenia DIR — w trybie ZX Spectrum i CP/J — czy XDIR w trybie CP/J). Jednakże pisząc odpowiednie procedury w kodzie maszynowym można czytać lub zapisywać dowolne obszary na dyskach Juniora, a także na dyskach zapisanych w innych formatach (np. 360 KB IBM)!

Struktura pamięci operacyjnej ELWRO 800-2 Junior

Chcąc wykonywać nietypowe operacje dyskowe należy poznać strukturę pamięci operacyjnej Juniora.

Komputer wyposażony jest w 64 KB pamięci RAM oraz 24 KB ROM.

Pamięć RAM zajmuje oczywiście pełny obszar przestrzeni adresowej procesora Z80. Pamięć ROM podzielona jest na dwa bloki: 16 KB głównej pamięci ROM zajmującej adresy od #0 do #3FFF, oraz dodatkowy ROM 8 KB w obszarze od #0 do #1FFF.

Jak więc widać w obszarze adresowym #0..#1FFF egzystują jednocześnie RAM i oba ROM-y, a w obszarze #2000..#3FFF — RAM i ROM podstawowy. Aby uniknąć kolizji jednoczesnego dostępu do różnych pamięci, Junior wyposażony jest w odpowiedni mechanizm dynamicznego przełączania różnych banków (stron) pamięci operacyjnej. Oczywiście normalnie użytkownik nie musi wiedzieć, który z bloków pamięci jest aktywny. Znowu robi to za niego system operacyjny. Dla dalszych rozważań należy jednak wiedzieć kiedy i które obszary pamięci są aktywne.

W trybie ZX Spectrum struktura pamięci jest następująca:

Główna pamięć RAM #4000..#FFFF (jak w ZX Spectrum). Obszar RAM #2000..#3FFF wykorzystywany jest przez system operacyjny (niewidocznie dla użytkownika) przy operacjach ze stacją dysków oraz operacjach sieciowych. Natomiast RAM od #0 do #1FFF w trybie ZX Spectrum jest nie wykorzystany.

Główny ROM 16 KB zawiera zmodyfikowany system operacyjny ZX Spectrum. Zmienione są np. komunikaty błędów, wzorce znaków, procedury obsługi drukarki, procedury obsługi przerwań (całkowicie zmieniono procedurę przerwania niemaskowalnego I, rozszerzono procedurę obsługi przerwania maskowalnego). Wykorzystano też ROM w obszarze #386E..#3CFF, który w ZX Spectrum był nie używany. Zasadnicze procedury obsługi magnetofonu są identyczne jak w ZX Spectrum. Jedynie na ich początku system sprawdza, czy chodzi w danym momencie o transmisję w sieci Junet, na dysk czy też na magnetofon. Zawartość głównego ROM-u Juniora można odczytywać przez np. PRINT PEEK adres oraz zdekodować korzystając z dowolnego deasemblera ZX Spectrum.

ROM dodatkowy 8 KB zawiera przede wszystkim procedury obsługi stacji dysków, procedury obsługi sieci komputerowej, oraz inne jak np. sprawdzanie konfiguracji komputera (ze stacją czy bez), odczyt swojego numeru komputera (każdy Junior ma zakodowany swój numer), wydruk na ekranie komunikatu początkowego po włączeniu zasilania Juniora lub po instrukcji NEW, itd. Normalnie ROM ten jest niedostępny dla użytkownika tzn. nie można go odczytać np. przez PRINT PEEK adres.

W trybie CP/J aktywna jest cała pamięć RAM 64 KB. W operacjach dyskowych system korzysta z procedur zawartych w ROM dodatkowym.

Mechanizm przełączania bloków pamięci

W Juniorze do przełączania bloków pamięci zarezerwowano jeden z możliwych adresów urządzeń wejścia-wyjścia. Wykonując instrukcję OUT (adres),A, przy czym adres = 247 (#F7) dokonywać można przełączania różnych bloków pamięci ROM i RAM. O tym, który blok pamięci ma zostać uaktywniony decyduje zawartość akumulatora A. Umiejętne wykorzystanie instrukcji OUT (#F7),A umożliwia także inne „sztuczki" jak np. możliwość sprawdzenia numeru komputera czy zmiana położenia ekranu w pamięci.

ELWRO 800 JUNIOR



Zbyt wiele złego napisano o mikrokomputerze edukacyjnym ELWRO 800 Junior, by nie wzbudzić zainteresowania tą „całkowicie oryginalną" (cytat ze wstępu do „Podręcznika użytkownika mikrokomputera ELWRO 800 Junior") konstrukcją rodzimych naukowców z Zakładu Badań Operacyjnych i Systemów Komputerowych Instytutu Automatyki Politechniki Poznańskiej.

Pierwsze spotkanie jest bardzo miłym rozczarowaniem. JUNIOR jawi się jako urządzenie o solidnej, budzącej zaufanie budowie. Całość utrzymana jest w konwencji czarno-białej, tzn. biała obudowa (przypomina coś, co można było nabyć w sklepach muzycznych) i czarna, wyraźnie opisana kontaktronowa klawiatura składająca się z trzech pól: alfanumerycznego (łącznie z polskimi znakami diakrytycznymi), funkcyjnego i pola kursorów. Obok znajduje się duża dioda sygnalizująca czerwonym kolorem pracę urządzenia. Włącznik sieciowy znajduje się z prawej strony obudowy, z tyłu zaś znajdujemy gniazda pozwalające na komunikację mikrokomputera ze światem zewnętrznym. Umożliwiają one przyłączenie monitora monochromatycznego, monitora kolorowego (RGB), magnetofonu kasetowego (gniazdo diodowe), drukarki, manipulatora drążkowego, pióra świetlnego i myszki; jako opcja przewidziane jest gniazdo do pamięci dyskowej 2*5,25". Ponadto są dwa gniazda do przyłączenia sieci JUNET, gniazdo do podłączenia sieci mikrokomputerów ZX Spectrum (w przypadku testowanych mikrokomputerów były one puste!) i duży przycisk RST (z ang. *RESET*) służący do sprzętowego zerowania mikrokomputera. Mimo zapowiedzi w instrukcji, brak jest gniazda umożliwiającego podłączenie odbiornika TV. Wszystkie gniazda są wyraźnie opisane i trudno je pomylić.

Po włączeniu zasilania na ekranie pojawia się oryginalna winieta (por. rysunek) informująca użytkownika o stanie zasobów mikrokomputera (24 KB pamięci ROM i 64 KB pamięci RAM) oraz o obecności (lub jej braku) stacji dysków. W tym stanie mikrokomputer pracuje pod kontrolą znajdującego się w pamięci ROM systemu (według nomenklatury instrukcji jest to „system rezydentny interpretera języka BASIC") w dużej mierze zgodnego z popularnym systemem znajdującym się w mikrokomputerze ZX Spectrum i jego pochodnych. Brak na klawiszach opisów charakterystycznych dla ZX Spectrum słów kluczowych wskazuje, że nie jest zgodne z zamysłem konstruktorów, by JUNIOR „udawał" ZX Spectrum. Raczej jest to wyjście ku tym, którzy zgromadzili dużą ilość programów na ten, tak w Polsce popularny, mikrokomputer (pracują one na JUNIORZE bez problemów, kłopoty mogą się pojawić przy współpracy z drukarką lub gdy program wykorzystuje przerwania). Jest to decyzja ze wszech miar słuszna. Tym, którzy koniecznie będą chcieli pisać programy w języku ZX BASIC proponuję używanie programu BetaBasic v.3.0., który — między innymi — oferuje opcję pozwalającą na wpisywanie słów kluczowych „literka po literce". Oryginalny system ZX Spectrum został uzupełniony o funkcje pozwalające na komunikowanie się mikrokomputerów między sobą (LOAD , SAVE , MERGE i DISPL , gdzie po znaku " " musi wystąpić adres mikrokomputera w sieci) i ze stacją dys-

Zawartości akumulatora i odpowiadające im warianty pamięci zawiera tabela I. Krótki program w asemblerze Z80 umożliwiający odczytanie numeru komputera pokazano na wydruku 1. Po zasemblowaniu i umieszczeniu kodu wynikowego w pamięci, wykonując PRINT USR 60000 otrzymamy na ekranie numer komputera, na którym pracujemy.

Niestety nie jest możliwe włączenie pełnej pamięci RAM z obrazem ZX Spectrum tzn. pod adresem #4000. Można natomiast wykonując w ZX Spectrum BASIC OUT 247,40 przełączyć ekran ZX Spectrum na CP/J (oczywiście dotyczy to jedynie wyświetlania obrazu — wszystkie procedury obsługi ekranu np. PRINT, CLS, itd., będą dalej działać tylko na ekranie ZX Spectrum).

Przełączanie bloków pamięci można oczywiście wykonać także innymi instrukcjami OUT procesora Z80, pamiętając jedynie, aby na mniej znaczącej części adresowej umieścić wartość F7. Możliwe jest również wykonanie w ZX Spectrum BASIC instrukcji OUT 247, argument, jednakże należy w tym przypadku zwrócić baczną uwagę na argument operacji OUT! W zasadzie możliwe jest jedynie wykonanie OUT 247,40 (zmiana ekranu) i OUT 247,168 (pełny RAM), przy czym uprzednio należy umieścić w odpowiednim miejscu RAM procedurę, która na końcu będzie posiadała instrukcję (oczywiście już w kodzie wewnętrznym Z80) OUT 247,8. Po jej wykonaniu nastąpi powrót do sytuacji wyjściowej (ROM podstawowy, RAM #4000..#FFFF). W wydruku 2 przedstawiono krótki program w asemblerze Z80, który umożliwia kopiowanie pamięci RAM #2000..#3FFF do RAM #8000..#9FFF w niecodzienny sposób przez OUT 247,168. Program po asemblacji należy uruchomić instrukcją (np.) RANDOMIZE USR 60000. Po uruchomieniu nastąpi wpisanie do pamięci RAM od adresu #1E7A odpowiedniego kodu. Procedura OUT w ZX Spectrum BASIC znajduje się dokładnie pod adresem #1E7F i w tym właśnie miejscu w pamięci RAM znajdzie się początek procedury kopiującej.

Program tam umieszczony nie ulega zniszczeniu nawet po wykonaniu sprzętowego kasowania!

Uwaga!

Nieostrożne wykonanie OUT 247 (szczególnie w ZX Spectrum BASIC, lecz również w kodzie wewnętrznym) z reguły powoduje unieruchomienie komputera!

Korzystając z mechanizmu przełączania bloków pamięci można przepisać odpowiednią procedurą w kodzie maszynowym zawartość ROM-u dodatkowego lub też RAM-u z obszarów #0..#3FFF do RAM-u w innym miejscu i następnie np. analizować procedury zawarte w ROM-ie dodatkowym.

Procedura w asemblerze Z80 pokazana na wydruku 3 przepisuje ROM dodatkowy do wolnej pamięci RAM.

Po wpisaniu powyższego programu za pomocą dowolnego asemblera dla ZX Spectrum, po wykonaniu asemblacji i umieszczeniu kodu wynikowego pod adresem 60000 d uruchomić go można przez (np.) RANDOMIZE USR 60000. W obszarze 32768..40960 (#8000..#9FFF) znajdzie się kopia ROM-u dodatkowego, którego zawartość można teraz analizować. Oczywiście program można umieścić w innym miejscu pamięci (inny argument dyrektywy ORG). Podobnie kopia ROM-u także może być umieszczona w innym miejscu (ADR0). Podstawiając za ADR1 zamiast #0 wartość #2000 możemy w ten sam sposób skopiować RAM z obszaru #2000..#3FFF.

W obszarze #2000..#3FFF pamięci RAM w trybie ZX Spectrum znajdują się bufory oraz zmienne systemowe używane przy operacjach dyskowych i sieciowych. Nie wykorzystany (w trybie ZX Spectrum) obszar RAM #0..#1FFF można wykorzystać jako RAM-dysk. Także obszar #2000..#3FFF można wykorzystać jako RAM-dysk, jeśli nie używa się pamięci dyskowej i sieci komputerowej. Można do tego wykorzystać analogiczne procedury jak powyżej, jedynie w linii 60 należy zmienić argument instrukcji LDA,#A8 (pełny RAM), a także ustawić odpowiednie adresy źródła i przeznaczenia danych oraz potrzebną długość bloku. Można w ten sposób np. zapamiętywać i wywoływać ekran nie zajmując przy tym głównej pamięci RAM.

Pamięć RAM #2000..#3FFF w operacjach dyskowych

Wszystkie systemowe operacje dyskowe Juniora w trybie ZX Spectrum wykorzystują obszar RAM #2000..#3FFF. Wykaz niektórych zmiennych i bloków tego obszaru zawiera tabela II. Ponadto w zmiennej systemowej nie wykorzystanej w ZX Spectrum pod adresem 23678 (#5CB0) bit 0 ustawiony informuje interpreter Junior BASIC o operacji dyskowej, a bit 1 o operacji sieciowej.

Procedury dyskowe zawarte w pamięci ROM

Niektóre z wielu procedur zawartych w ROM-ie dodatkowym (8 KB), które mogą być wykorzystane przez użytkowników we własnych programach w kodzie wewnętrznym Z80 zamieszczone są w tabeli III.

---

ków (LOAD*, SAVE* i MERGE*). Możliwe jest również wyprowadzenie na ekran katalogu dysku (DIR lub DIR nr stacji, co równocześnie czyni wskazany dysk aktywnym) i przywołanie właściwego dla JUNIORA systemu dyskowego CP/J.

Jak już wspomniano, bez stacji dysków elastycznych i organizacji sieci JUNET, JUNIOR jest tylko rozbudowaną wersją ZX Spectrum. Prawdziwe oblicze odkrywa, gdy przy pełnej konfiguracji wprowadzi się zlecenie CP/J. Powoduje to odłączenie pamięci ROM na rzecz pamięci RAM i... otrzymujemy zestaw skomunikowanych ze sobą mikrokomputerów pracujących pod kontrolą systemu zgodnego z najpopularniejszym na świecie dyskowym systemem operacyjnym dla ośmiobitowych mikrokomputerów, jakim jest CP/M 2.2 (ang. *Control Program/Monitor*). Komunikację między mikrokomputerami połączonymi w sieć oraz stacją dysków zapewnia program nadzorujący o nazwie JUNET rezydujący w wyróżnionym mikrokomputerze, zwanym nauczycielskim. Sieć tworzy mikrokomputer nauczycielski z przyłączoną stacją dysków elastycznych i do piętnastu mikrokomputerów (tzw. uczniowskich), niektóre z nich mogą być również wyposażone w indywidualne stacje dysków elastycznych (o ile wyposażone są w moduły sterowników dysków).

Każdy z uczniów ma dostęp do stacji dysków przyłączonej do mikrokomputera nauczycielskiego, gdzie znajdują się dwojakiego rodzaju zbiory: typu PUBLIC, ■ więc ogólnie dostępne (np. programy użytkowe), ale zabezpieczone przed dokonywaniem zmian; i typu LOCAL, czyli zbiory dostępne jedynie przez wskazany mikrokomputer (zwykle zbiory kreowane przez ucznia) i możliwe do dalszej modyfikacji. Tak więc każdy z uczniów ma swój prywatny katalog różny od sąsiadów mimo korzystania z tej samej stacji dysków. Skutecznie chroni to przed kolizjami przy dostępie do zbiorów.

Operator mikrokomputera nauczycielskiego ma możliwość pełnej kontroli pracy uczniów. Przede wszystkim bez ograniczeń dostępne są wszelkie zbiory na dyskach. Nauczyciel może kontrolować katalog dyskietki, jak i katalog dowolnie wybranego jej użytkownika (czyli ucznia), może przeglądać zawartości wybranych plików i je drukować. Dalej przewidziano możliwość podglądu ekranu monitora wskazanego ucznia oraz wysyłania do uczniów komunikatów. Można wreszcie rozesłać do określonej grupy uczniów wybrany program (tu ograniczenie: objętość rozsyłanego programu nie może przekraczać 32 KB) lub system operacyjny CP/J do tych mikrokomputerów, w których nie był on zainstalowany.

Adresowanie mikrokomputerów uczniowskich odbywa się za pomocą adresów sieciowych lub identyfikatorów. Każdemu użytkownikowi (0..15) przyporządkowany jest jednoznacznie jego adres sieciowy (16..63), dokonuje się tego dla konkretnej sieci raz za pomocą zlecenia INSTAL. Oprócz tego (dla danej grupy użytkowników, np. klasy) można użytkownikom przypisać identyfikatory (np. nazwiska), co czyni komunikację z uczniami bardziej naturalną. Każda klasa może mieć swój indywidualny zbiór identyfikatorów, które przechowywane są na dysku elastycznym, i mogą być wprowadzone do pamięci mikrokomputera nauczycielskiego podczas zarządzania siecią JUNET. Wydaje się jednak, że to potrójne przyporządkowanie (przy rozsyłaniu programów trzeba posługiwać się adresami grupowymi) może niektórym nauczycielom sprawiać trudności. Obsługa programu nadzorującego sieć JUNET nie stwarza większych problemów, choć niektóre komunikaty są zbyt lakoniczne, pojawiają się w dziwnych miejscach, ■ nawet można doprowadzić przez nieuwagę do dezorganizacji pracy mikrokomputera nauczycielskiego.

Na dołączanym do sieci systemowym dysku oprócz systemu CP/J, procedur składających się na zestaw zleceń nierezydentnych systemu i procedury JUNET

Komunikacja systemu ze sterownikiem stacji

„Pośrednikiem" pomiędzy mikrokomputerem Junior a pamięcią dyskową jest układ scalonego sterownika stacji dysków elastycznych NEC μPD 765 (odpowiednik INTEL 8272) — układ ten jest również stosowany w komputerach IBM PC XT/AT. Opis tego układu został zamieszczony w numerze 1/1988 czasopisma „Mikroklan". Układ μPD 765 może pracować w dwóch trybach — tzw. DMA (bezpośredni zapis-odczyt pamięci z pominięciem procesora komputera) oraz NON-DMA (procesor zapisuje i odczytuje dane ze sterownika). W Juniorze wybrano drugi tryb.

Procesor Z80 komunikuje się ze sterownikiem za pośrednictwem trzech portów: #EF, #EE oraz #F1.

Przez OUT (#F1),A (A musi posiadać odpowiednią zawartość), procesor włącza i wyłącza jeden z dwóch napędów stacji (stacja Juniora jest podwójna).

Przez port #EF procesor wysyła do sterownika (OUT) kody poleceń i dane, odbiera natomiast dane odczytane z dysku (IN).

Przez port #EE procesor odczytuje (IN) ze sterownika jego główny rejestr stanu. W rejestrze stanu najważniejszy jest stan bitu 7, który informuje (po uaktywnieniu jednego z dwóch napędów przez OUT (#F1),A o gotowości sterownika i stacji do działania.

Przykłady niestandardowych procedur obsługi stacji

Wydruki 4 i 5 przedstawiają dwa przykłady procedur zapisu-odczytu pamięci dyskowej Juniora.



Pierwsza z nich (wydruk 4) pozwala na odczytanie lub zapisanie dowolnego bloku na dysku (umożliwia np. bezpośredni odczyt katalogu dysku, odczyt pierwszych bloków typu ZX Spectrum, w których — pierwsze 17 bajtów — zawarty jest nagłówek zbioru), a także czytanie w trybie ZX Spectrum zawartości zbiorów typu „nie-Spectrum" (CP/J).

Druga (por. wydruk 5) umożliwia odczyt i zapis dowolnego sektora. Za jej pomocą można odczytywać normalnie ukryte dla użytkownika ścieżki systemowe (ścieżka 0 na stronach 0 i 1)! Ponadto pozwala na odczyt (i zapis) dysków zapisanych w innych formatach niż Junior!

W procedurze odczytu-zapisu bloku (wydruk 4) w komórce NRDYS (60040) należy umieścić numer napędu stacji (0 lub 1). W komórce KODOP (60041) należy wpisać 4, jeśli chcemy odczytać blok, lub też 6, jeśli chcemy zapisać blok. Dwie komórki NRBLK (60043) powinny zawierać numer bloku, który zamierzamy odczytać lub zapisać.

W komórkach ADRES wpisujemy adres początku obszaru pamięci, do którego zostaną wpisane dane odczytane z dysku przy odczycie bloku (lub z którego zostaną pobrane dane do zapisania na dysku przy zapisie bloku). W przykładzie podano adres #8000 (32768), a więc odczyt-zapis będzie dotyczył obszaru od 32768 do 34815.

Procedurę wywołuje się przez np. PRINT USR 60000.

---

wraz z plikami roboczymi znajduje się szereg cennych programów roboczych jak: makroasemblery MBO i MAC, program pomocniczy CREF80 do MBO pozwalający na uzyskiwanie tekstu źródłowego poszerzonego o dodatkowe informacje, program łączący L80, dwa (SID i ZSID) programy pomocne przy uruchamianiu procedur w języku asemblera (ang. *debugger*). Komplet ten uzupełnia pełnoekranowy edytor przeznaczony do redagowania programów źródłowych o nazwie EDJ (jest on zgodny z regułami wprowadzonymi przez firmę Borland International wraz z edytorem tekstów WordStar). Ten elementarny zestaw dopełnia kompilator języka Pascal (zgodny z Turbo Pascalem wersja 3.0 firmy Borland International) wyposażony w automatyczny edytor (te same zasady co przy edytorze EDJ) wraz z pakietem elementarnych procedur graficznych oraz interpreter języka BASIC (implementacja języka BASIC-80). Do każdej z wymienionych pozycji producent załącza wyczerpującą (choć chwilami trudną w czytaniu) instrukcję w języku polskim. Dodać należy, że od kilku miesięcy dostępny jest również interpreter języka LOGO tak w wersji angielskiej, jak i polskiej (tzw. PTI-LOGO). Jak na młody wiek JUNIOR-a to wcale pokaźny zestaw i winien być podmiotem oddzielnych testów.

Po blisko półrocznym intensywnym testowaniu sieci JUNIOR-ów złożonej z siedmiu mikrokomputerów (1+6) i stacji dysków elastycznych można z zadowoleniem stwierdzić, że w dużej mierze spełnia ona pokładane nadzieje. Nie ziściły się kasandryczne doniesienia o beznadziejnej pracy stacji dysków elastycznych (dotyczyły wcześniejszego typu) i częstych błędach przy transmisji danych w sieci JUNET.

Poniżej zawarte jest krótkie porównanie parametrów JUNIORA.

Zalety

- możliwość pracy w sieci i zgodność z CP/M,
- dostępne z klawiatury polskie znaki diakrytyczne oraz (w trybie ZX Spectrum) komunikaty w języku polskim,
- zwartość budowy (zasilacz w środku obudowy),
- trwała klawiatura,
- duża (blisko 1.4 MB) pojemność dostępnej pamięci dyskowej,
- możliwość wykorzystania bogatej biblioteki oprogramowania mikrokomputera ZX Spectrum,
- prosta obsługa komunikacji ze stacją dysków elastycznych w systemie ZX Spectrum.

Wady

- brak dostępnego z zewnątrz bezpiecznika,
- głośna praca stacji dysków elastycznych,
- brak możliwości rozsyłania programów do dowolnie wybranego mikrokomputera (za pomocą adresu indywidualnego lub identyfikatora),
- brak programów systemowych (w CP/J) pozwalających kopiować zbiory z dysków o mniejszych pojemnościach (standardy CP/M),
- brak oprogramowania dydaktycznego pracującego pod nadzorem systemu CP/J (miejmy nadzieję, że chwilowo),
- wysokie wymagania dotyczące elektrycznej sieci zasilającej,
- brak wyprowadzenia odpowiednio buforowanej magistrali systemowej,
- zdaniem entuzjastów ZX Spectrum, brak opisu słów kluczowych ZX Spectrum na klawiaturze,
- brak możliwości przerwania (przez wykonanie BREAK) działania instrukcji LPRINT, LLIST i COPY (w trybie ZX Spectrum),
- brak gniazda umożliwiającego podłączenie odbiornika TV.

TADEUSZ A. ZALESKI

Wydruk 1. Procedura odczytu numeru komputera

```
0010 ;
0020 ;                     ;"Odczyt numeru komputera"
0060 ;
0070       ORG 60000       ;adres wywołania
0080 ;
0090 START DI              ;zablokuj przerwania
0100       LD A,10H        ;przygotowanie do odczytu
0110       OUT (0F7H),A    ;   numeru
0120       LD A,0FFH       ;
0130       IN A,(0FEH)     ;A:= numer komputera
0140       LD B,0          ;
0150       CPL             ;negacja akumulatora
0160       AND 3FH         ;weź tylko bity 0..5
0170       LD C,A          ;C - numer komputera
0180       LD A,8          ;ROM pod., RAM 16..64 KB
0190       OUT (0F7H),A    ;przełącz pamięć
0200       EI              ;odblokuj przerwania
0210       RET             ;powrót - BC nr komp.
0220       END
```

Wydruk 2. Procedura kopiowania RAM przez OUT 247,168

```
0010 ;
0020 ;                     ;"kopiowanie RAM 2000..3FFF do RAM
0030 ;                     ;" 8000..9FFF"
0040 ;                     ;"kopiowanie przez OUT 247,168 w
0050 ;                     ; ZX Spectrum BASIC"
0060 ;
0070       ORG 60000       ;adres wywołania 60000
0080 ;
0090 DLUG  EQU 20H         ;długość bloku kodu
0100 KOPY3 EQU 1E7AH       ;adres przeznaczenia bloku kodu
0110 ;
0120 KOPY1 DI              ;zablokuj przerwania
0130       LD HL,KOPY2     ;HL:=adres źródła danych
0140       LD DE,KOPY3     ;DE:=adres przeznaczenia danych
0150       LD BC,DLUG      ;BC:=długość bloku
0160       LD A,0A8H       ;włącz pełny RAM
0170       OUT (0F7H),A    ;
0180       LDIR            ;przeładuj pamięć
0190       LD A,0C9H       ;A:=kod RET i wpisz do
0200       LD (38H),A      ; adresu INT oraz
0210       LD (66H),A      ; NMI
0220       LD A,8          ;ROM pod., RAM 16..64 KB
0230       OUT (0F7H),A    ;przełącz pamięć
0240       EI              ;odblokuj przerwania
0250       RET             ;powrót do ZX Spectrum BASIC
0260 ;
0270 KOPY2 LD A,8          ;A - przygotow. do przełącz.
0280       EI              ;odblokuj przerwania
0290       OUT (0F7H),A    ;wróć do BASIC (ROM pod.adres 1E7F)
0300 ;
0310 S1E7F DI              ;tu wejście po OUT 247,168 (BASIC)
0320       PUSH HL         ;zabezpiecz rejestry
0330       PUSH DE         ;
0340       PUSH BC         ;
0350       LD BC,2000H     ;BC:=dł. bloku
0360       LD HL,2000H     ;HL:=adres bloku
0370       LD DE,8000H     ;DE:=adres przeznaczenia
0380       LDIR            ;przeładuj  RAM  2000..3FFFh  do
                           ; 8000..9FFF hex
0390       POP BC          ;odtwórz rejestry
0400       POP DE          ;
0410       POP HL          ;
0420       JR KOPY2        ;skok do wyjscia
0430       END
```

Wydruk 3. Procedura kopiowania ROM dod. - RAM 8000..9FFFh

```
0000       ORG 60000       ;adres proc. 60000 dec.
0010 ;
0020 ADR0  EQU 32768       ;adres przeznacz.danych
0030 ADR1  EQU 0           ;adres źródła danych
0040 ;
0050 COPY  DI              ;zablokuj przerwania
0060       LD A,48H        ;akumulator :=48 hex
0070       OUT (0F7H),A    ;włącz ROM dodatk.
0080       LD HL,ADR1      ;HL:= adres źródła danych
0090       LD DE,ADR0      ;DE:= adres przeznaczenia
0100       LD BC,2000H     ;BC:= długość bloku danych
0110       LDIR            ;przepisz blok dł.8192 dec
0120       LD A,8          ;akumulator :=8
0130       OUT (0F7H),A    ;ROM podst.,RAM 16..64 KB
0140       EI              ;odblokuj przerwania
0150       RET             ;powrót
0160 ;
0170       END
```

Wydruk 4. Procedura odczytu-zapisu jednego bloku.

```
0010 ;
0020 ;
0030       ORG 60000       ;adres wywoł.procedury
0040 ;
0050 BLOK  EQU 0ACDH       ;adres proc.zapisu-odczytu
0060 DYSNR EQU 3209H       ;adres zmien. - nr FDD
0070 ;
0080 RWBLK DI              ;zablokuj przerwania
0090       PUSH IY         ;przechowaj IY na stosie
0100       LD A,48H        ;A:=48 hex
0110       OUT (0F7H),A    ;włącz ROM dod. i RAM 8..64 KB
0120       LD A,(NRDYS)    ;A:=numer FDD
0130       LD (DYSNR),A    ;i wpisz do 3209h
0140       LD BC,(KODOP)   ;BC:= kod op.4-odcz.,6-zapis
0150       PUSH BC         ;i umieść na stosie
0160       LD BC,(NRBLK)   ;BC:=numer bloku
0170       LD DE,(ADRES)   ;DE:=ad.zapisu-odczytu bloku
0180       CALL BLOK       ;czytaj-zapisz blok
0190       LD B,0          ;BC:=kod błędu 255 dobrze
0200       LD C,A          ;     0 - źle
0210       LD A,8          ;ROM podst.,RAM 16..64 KB
0220       OUT (0F7H),A    ;
0230       POP IY          ;odtwórz rej. IY
0240       EI              ;odblokuj przerwania
0250       RET             ;wróć do BASIC ZX Spectrum
0260 ;
0270 NRDYS DEFB 0          ;nr FDD (0 lub 1)
0280 KODOP DEFW 4          ;kod oper. (4-load, 6-save)
0290 NRBLK DEFW 0          ;numer bloku
0300 ADRES DEFW 8000H      ;adres  początku  bloku  w
                           ;pamięci
0310 ;
0320       END
```

Wydruk 5. Procedura odczytu/zapisu jednego sektora.

```
0020       ORG 60000       ;adres wywołania
0030 ;
0040       DI              ;zablokuj przerwania
0050       PUSH IY         ;IY na stos
0060       LD A,48H        ;ROM dod.,  8..16 KB
0070       OUT (0F7H),A    ;przełącz pamięć
0080       LD DE,2009H     ;DE:=adr. kodów ster.
0090       LD HL,USTAW     ;uPD 765 - tu ustaw.głow.
0100       LD BC,3         ;na ścieżkę
0110       LDIR            ;przepisz kody
0120       LD A,(NRDYS)    ;A:=numer FDD
0130       CALL 01EEH      ;włącz FDD
0140 WAIT  IN A,(0EEH)     ;odcz. rej. stat. sterow.
0150       BIT 7,A         ;czy stacja gotowa ?
0160       JR Z,WAIT       ;skok jeśli nie gotowa
0170       CALL OPOZN      ;opóźnienie
0180       LD BC,2007H     ;BC:=adr. kodów sterown.
0190       CALL 99AH       ;ustaw głowicę na ścieżkę
0200       LD DE,2007H     ;wpisz 2007..2010h kody dla
0210       LD HL,RDWR      ; sterownika
0220       LD BC,0BH       ;BC:=dług.bloku kodów
0230       LDIR            ;przepisz
0240       CALL OPOZN      ;opóźnienie
0250       LD BC,2007H     ;BC:=adres kodów sterownika
0260       CALL 99AH       ;zapisz/odczytaj sektor
0270       CALL OPOZN      ;opóźnienie
```

Przy prawidłowym odczycie lub zapisie na ekranie powinniśmy otrzymać 255. Jeśli odczyt lub zapis nie był prawidłowy (np. dysk w innym formacie) na ekranie pojawi się 0.

Procedura z wydruku 5 umożliwia odczyt lub zapis dowolnego sektora na dysku formatu 720 KB, jak również w innych formatach np. 360 KB IBM. Wywołana z takimi jak podano w wydruku parametrami odczytuje pierwszy sektor ścieżki 0 na stronie 0 dysku umieszczonego w stacji 0 (czyli A) — w formatach 720 KB, 360 KB i 180 KB (jednostronne). Dane odczytane z dysku wpisuje do RAM od 32768 do 33279 (#8000..#81FF).

W celu odczytania innych sektorów należy zmienić poszczególne parametry dla sterownika μPD 765.

W komórce NRDYS można umieścić 0 lub 1 (napęd A lub B).

W komórkach STDY1 i STDY2 należy podać stronę i numer napędu: bity 0 i 1 — nr napędu od 0 do 3 (w Juniorze 0 lub 1), bit 2 — strona 0 lub 1.

W komórkach SCIE1 i SCIE2 należy wpisać numer ścieżki (od 0 do 79), przy czym nie muszą to być te same wartości. Jeśli do stacji włożymy dysk np. 360 KB (każda strona zawiera tu po 40 ścieżek, a więc dwa razy mniej niż Junior) to chcąc odczytać jakiś sektor ze ścieżki np. 10, należy wpisać do SCIE1 wartość 20, a do SCIE2 wartość 10. Inaczej mówiąc należy ustawić głowicę stacji na odczyt ścieżki 20, a odczytać 10.

W komórce STR należy jeszcze raz podać numer strony dysku (0..1), a w komórce SEKT umieścić numer sektora 1..9.

Od adresu USTAW (#EAB = 60086) umieszczone są dane dla sterownika μPD 765:

kod instrukcji „ustaw_głowicę_napędu_na_ścieżkę" (1 komórka — #0F), strona i nr napędu (1 komórka), numer ścieżki (1 komórka).

Od adresu RDWR umieszczone są dane dla właściwej operacji zapisu lub odczytu sektora:

```
0280        CALL 1F8H      ;wyłącz FDD
0290        LD A,(201AH)   ;kod błędu do BC
0300        LD C,A         ;
0310        POP IY         ;odtwórz IY
0320        LD A,8         ;ROM podst., [illegible] 16..64 [illegible]
0330        OUT (0F7H),A   ;przełącz pamięć
0340        EI             ;odblokuj przerwania
0350        RET            ;wróć do BASIC ZX Spectrum
0360 ;
0370 OPOZN  LD BC,80H      ;procedura opóźni@jąca
0380 LOOP   DJNZ LOOP      ;
0390        DEC C          ;
0400        JR NZ,LOOP     ;
0410        RET            ;powrót do głównej proc.
0420 ,
0430 USTAW  DEFB 0FH       ;kod ster. - ustaw głowicę
0440 STDY1  DEFB [illegible]        ;strona dysku, nr FDD
0450 SCIE1  DEFB 0         ;numer ścieżki (0..79)
0460 ;
0470 RDWR   DEFW 8000H     ;adres zapisu-odczytu
0480 CODE   DEFB 46H       ;kod ster.-tu odczyt sekt.
0490 STDY2  DEFB 0         ;strona dysku, nr FDD
0500 SCIE2  DEFB 0         ;nr ścieżki (0..79)
0510 STR    DEFB 0         ;strona (0..1)
0520 SEKT   DEFB 1         ;nr sektora (1..9)
0530        DEFB 2         ;512 bajtów/sektor
0540        DEFB 9         ;9 sektorów/ścieżkę
0550        DEFB 12H       ;odstęp między sektorami
0560        DEFB 0FFH      ;stała 255
0570 NRDYS  DEFB 0         ;nr FDD (0 lub 1)
0580        END
```

Wydruk 6. Program czytania/zapisu sektorów w formatach 720 i 360 KB.

```
10 LOAD *"CSEKTOR"CODE 60000,101
20 CLS
30 INPUT "DYSK 1/0";D: IF D<>0 AND D<>1 THEN GO TO 30
40 INPUT "OPERACJA L/S";K$: IF K$<>"L" AND K$<>"S" THEN  GO
   TO 40
50 IF K$="L" THEN POKE L+91,70
60 IF K$="S" THEN POKE L+91,69
70 INPUT  "FORMAT 720/360 KB" (1/2";FORM: IF FORM  <>1  AND
   FORM<>2 THEN GO TO 70
80 INPUT "SEKTOR 0..1439/0..719";NRSE: IF NRSE*FORM>1439
   THEN GO TO 80
90 REM ***    numer sektora na ścieżce (0..8)
100 LET SEK=NRSE-9*INT (NRSE/9)
110 REM ***    numer ścieżki
120 LET SCI=INT ((NRSE-SEK)/18)
130 REM ***    strona
140 LET STR=(NRSE-SEK)/9-SCI*2
150 REM ***    sektor 1..9
160 LET SEK=SEK+1
170 REM ***
180 LET L=60000
190 REM ***    strona i numer dysku
200 POKE L+87,D+4*STR: POKE L+92,PEEK (L+87)
210 REM ***    strona
220 POKE L+94,STR
230 REM ***    ścieżka (*2 dla 360 KB)
240 POKE L+88,SCI*FORM: POKE L+93,SCI
250 REM ***    sektor
260 POKE L+95,SEK
270 REM ***    numer dysku
280 POKE L+100,D
290 REM ***
300 LET M=USR L: CLS : IF M<>0 THEN PRINT "BŁAD ZAPISU-
    ODCZYTU": GO TO 30
310 CLS : PRINT "SEKTOR ";NRSE,"FORMAT ";FORM: PRINT
320 LET ADRES=32768: FOR F=ADRES TO ADRES+511
330 PRINT F;"  ";PEEK F,CHR$ (PEEK F AND PEEK F >31)
340 NEXT F
350 GOTO 20
```

Uwaga: w linii 10 zbiór "CSEKTOR" to kod wynikowy uzyskany po zasemblowaniu procedury z wydruku 5.

TABELA I

1. #08 (8) — #0..#3FFF ROM podst., #4000..#FFFF RAM, ekran ZX Spectrum (#4000).
2. #28 (40) — #0..3FFF ROM podst., #4000..#FFFF RAM, ekran CP/J (#E000).
3. #48 (72) — #0..1FFF ROM dodatk., #2000..#FFFF RAM, ekran ZX Spectrum (#4000).
4. #68 (104) — #0..#1FFF ROM dodatk., #2000..#FFFF RAM, ekran CP/J (#E000).
5. #88 (136) — #0..#1FFF ROM podst., #2000..#FFFF RAM, ekran ZX Spectrum (#4000).
6. #A8 (168) — #0..#FFFF RAM, ekran CP/J (#E000).
7. #C8 (200) — #0..#1FFF ROM dodatk., #2000..#3FFF ROM podst., #4000..#FFFF RAM, ekran ZX Spectrum (#4000).
8. #E8 (232) — #0..#1FFF ROM dodatk., #2000..#3FFF ROM podst., #4000..#FFFF RAM, ekran CP/J (#E000).
9. #10 (16) — umożliwia odczyt nr komputera.

TABELA II

1. #2007 — adres odczytu lub zapisu jednego sektora.
2. #2009..#2011 — kody instrukcji i dane dla układu scalonego sterownika stacji NEC μPD 765.
3. #2012..#201A — kody błędów sterownika dysków.
4. #2080..#28FF — bufor, do którego wczytywane są bloki katalogu dysku (bloki 0..3). W danym momencie wpisany tam może być oczywiście tylko jeden blok (2 KB).
5. #29EC..#29FC — nagłówek zbioru typu ZX Spectrum.
6. #29FD..#32FC — bufor, do którego wczytywane są z dysku (lub z którego zapisywane są na dysk) poszczególne bloki danych pliku. W pierwszych blokach plików typu ZX Spectrum pierwsze 17 bajtów zawiera nagłówek zbioru.
7. #3209 — numer aktywnej stacji dysków.
8. #3800..#3EAF — bufor, w którym system umieszcza przygotowany do wydruku na ekranie katalog dysku po wykonaniu polecenia DIR (w trybie ZX Spectrum).

TABELA III

1. #01ED — włączenie stacji. Numer napędu w rej. C (#01EE — numer w akumulatorze). Numer napędu zapamiętywany jest w rej. IY.
2. #01F8 — wyłączenie napędu. Nie wymaga parametrów, gdyż numer stacji pamiętany jest w rejestrze IY. W czasie od włączenia (proc. #01ED) do wyłączenia stacji (proc. #01F8) nie wolno używać procedury #01ED.
3. #099A — elementarna procedura komunikacji systemu ze scalonym sterownikiem stacji NEC μPD 765. Po umieszczeniu w odpowiednim miejscu RAM kodu instrukcji oraz danych dla sterownika stacji (normalnie pod adresem #2009), wpisaniu w dwóch poprzednich komórkach (normalnie #2007) adresu zapisu lub odczytu sektora, należy w parze rejestrów BC podać adres (standardowo #2007) ww. parametrów i wywołać procedurę przez CALL #099A. W komórkach pamięci od #2011..#201A wpisywane są kody dotyczące statusu sterownika (najważniejsza jest zawartość #201A).
4. #0ACD — podstawowa procedura zapisu-odczytu jednego bloku (2 KB). Przed jej użyciem należy umieścić na stosie procesora kod operacji np. ładowania bloku — LD BC, 0004, PUSH BC (zapis bloku — LD BC.0006. PUSH BC), następnie w parze rejestrów BC należy umieścić numer bloku, a w parze DE adres zapisu-odczytu bloku.

— adres odczytu (zapisu) sektora (2 komórki), kod operacji „odczytaj_sektor" (46) (lub „zapisz_sektor" — 45), strona i nr napędu (1 komórka), numer ścieżki (1 komórka), strona (1 kom.), numer sektora (1 kom.). Dalej ilość 256-bajtowych bloków danych w jednym sektorze, liczba sektorów na ścieżkę, długość przerwy między sektorami, liczba 255, gdy sektor zawiera więcej niż 256 bajtów danych.

Dane od adresu SCIE2 do bajtu zawierającego #FF muszą być przy odczycie lub zapisie sektora identyczne z tzw. identyfikatorem sektora, który umieszczany jest na początku każdego sektora podczas formatowania dysku. Jeżeli dysk jest inaczej sformatowany np. jedna ścieżka zawiera 8 sektorów, to chcąc go odczytać należy zmienić niektóre dane dotyczące identyfikatora sektora (w tym przypadku liczbę sektorów na ścieżkę, a możliwe, że także i inne np. odstęp między sektorami, długość sektora).

Aby ułatwić posługiwanie się przedstawioną procedurą przy odczycie-zapisie dysków w dwóch formatach 720 KB i 360 KB można napisać dodatkowo program, który po określeniu numeru sektora (od 0 do 1439 dla dysku 720 KB, lub od 0 do 719 dla 360 KB) wyliczy numer strony, numer ścieżki i numer sektora na ścieżce. Dla łatwiejszego zrozumienia program napisany jest w ZX Spectrum BASIC (por. wydruk 6).

Po uruchomieniu programu należy podać nr napędu, rodzaj operacji (*Load-Save*), format dysku oraz numer sektora.

W liniach 100..160 program przelicza numer sektora (0..1439 lub 0..719) na potrzebne dane tzn.: numer sektora na ścieżce, numer ścieżki i stronę. Następnie od linii 200 do 280 wpisuje wyliczone dane do odpowiednich komórek pamięci, gdzie stanowią one dane dla sterownika dysków. Po pomyślnym zakończeniu procedury maszynowej (linia 300) zmienna M = 0 i następuje wydruk na ekranie odczytanych (zapisanych) danych. W przeciwnym wypadku sygnalizowany jest błąd.

Program na wydruku 6 jest bardzo uproszczony. Można go rozbudować o wiele

różnych funkcji, przede wszystkim, jeśli chodzi o wydruk (np. szesnastkowo). Można także przenieść np. przeliczenia sektorów do procedury maszynowej, itd.

Powyższy tekst został w całości napisany na mikrokomputerze Junior pracującym w trybie CP/J za pomocą edytora tekstów EDJ znajdującego się na dysku firmowym.

Procedury maszynowe napisane zostały przy użyciu asemblera EDITAS firmy Picturesque komputera ZX Spectrum (asembler przystosowano do współpracy ze stacją dysków Juniora).

Następnie przy użyciu programu kopiującego napisanego w całości w asemblerze (znowu EDITAS) plik z niniejszym tekstem został przeniesiony na dysk w formacie IBM 360 KB.

Po zamianie kodów polskich znaków diakrytycznych Juniora na kody „odpowiednie" dla IBM-a (patrz INFORMIK nr 2/1988 str. 11) — to już wykonał krótki program w Pascalu na IBM AT — tekst został „uformowany" pod kontrolą edytora WORDSTAR v.4.00 (na komputerze IBM AT) i wydrukowany na drukarce STAR NL 10.

Od redakcji: *Ze względów poglądowych w tekście artykułu przyjęto konwencję zapisu liczb szesnastkowych (heksadecymalnych) jako ciągu znaków poprzedzonych znakiem #. W treści procedur zastosowany jest format zapisu żądany przez asembler, tzn. liczba heksadecymalna przedstawiana jest jako ciąg znaków zaczynający się od cyfry i kończący literą H. Pozostałe liczby są liczbami w systemie dziesiętnym.*

PAKIET PROGRAMÓW
„Czytaj i pisz po angielsku"

„Czytaj i pisz po angielsku" to tytuł pakietu dziesięciu programów edukacyjnych wydanych przez Krajowe Wydawnictwo Czasopism RSW „Prasa-Książka-Ruch" i Redakcję Programów Komputerowych jako 17 pozycja programów komputerowych dostępnych na kasetach magnetofonowych. Pakiet ten stanowić ma pomoc w nauce języka angielskiego dla dzieci w wieku od 5 do 9 lat. Przeznaczony jest na komputer ZX Spectrum (i zgodne z nim) i składa się z pięciu kaset (na każdej po dwa programy). Cena pakietu 3500 zł. Programy te są polskimi adaptacjami wydanych w 1984 roku przez firmę Macmillan Education programów edukacyjnych pod nazwą **Learn to Read**.

Zanim przejdziemy do oceny walorów edukacyjnych programów zatrzymajmy się na chwilę nad ich cechami dającymi się określić jako komercyjne.

- Problematyczne wydaje się wydanie pakietu na pięciu kasetach. Motywem podjęcia takiej decyzji było z pewnością ułatwienie użytkownikowi wyszukania konkretnego programu, jednak rozwiązanie to znacznie podniosło cenę zestawu. Należałoby rozważyć możliwość nagrania wszystkich programów na jednej kasecie i sprzedaży zestawu po niższej cenie.



- Dyskusyjna jest forma graficzna winiety programu (uwaga ta odnosi się do większości kaset z programami wydanymi przez KWCz). Opracowana bezsprzecznie starannie razi pewną topornością (winieta nie zawiera bezpośredniego wyszczególnienia nazwiska grafika).
- Bezsprzecznie naganny jest niechlujny sposób dystrybucji kaset. Dostarczane są one luzem (nawet bez zbiorczego kartonowego pudełka), kolory pudełek kaset różnią się od siebie, winiety są niedopasowane do pudełek i wystają poza ich obręb.
- Jakość kaset można określić jako przeciętną (na kasetach umieszczony jest znak firmowy WIFONU). Zastosowana do nagrania programu Szybka Pamięć Taśmowa (autorstwa p. A. Pucka) skutecznie skraca czas ładowania, a dobre nagranie wysokim poziomem powoduje, że praktycznie nie występują kłopoty z wprowadzeniem programu do komputera.
- Instrukcja programu zawarta jest na winietach kaset. Jest bardzo lakoniczna, chwilami aż do niezrozumiałości. Wobec poważnej roli, jaką ma pełnić program edukacyjny, takie skrótowe potraktowanie instrukcji jest niedopuszczalne. Wydaje się, że jeżeli z takich czy innych przyczyn nie można było dołączyć do pakietu porządnie opracowanej instrukcji w postaci książeczki, to należało ją załączyć w postaci pliku tekstowego z programem udostępniającym na ekranie komputera. Na marginesie należy dodać, że nawet tak skrótowo potraktowana instrukcja zawiera błędy i nieścisłości (brak opisu lekcji 6 — zamiast niej powielono opis lekcji 7, w opisie lekcji 8 występują rozbieżności w nazewnictwie w stosunku do programu).

Rozpatrzmy teraz walory edukacyjne zestawu. Programy te w założeniu przeznaczone były do nauki pisania i czytania dla dzieci angielskich, i stanowiły niewielki procent całej gamy analogicznych programów edukacyjnych (jak np. Verbs, Flower, Anagram, Idioms i wiele innych). Przeznaczenie ich na komputer ZX Spectrum podyktował okres ich powstania (pamiętajmy, że jako komputer dla szkolnictwa brytyjskiego wybrany został BBC Acorn). Różnice w stosunku do oryginałów sprowadzają się jedynie do poleceń i niektórych opisów, które zostały zastąpione polskimi. Testując program usiłowałem wczuć się najpierw w rolę małego Polaka uczącego się języka angielskiego (posługując się wersją spolszczoną), a następnie w rolę małego Anglika uczącego się czytać i pisać (posługując się wersją oryginalną). Od komputera odchodziłem z mieszanymi uczuciami — odniosłem wrażenie, że ani z jednego, ani z drugiego punktu widzenia program nie spełnia swoich założeń.

Podobne opinie usłyszałem też, niestety, od młodych ludzi, którym udostępniłem kasety do przetestowania. Podstawowe zarzuty, które postawiłbym programom z punktu widzenia dydaktyki polskiej są następujące (opinia uwzględnia doświadczenia własne i zaprzyjaźnionej młodzieży):

- Nieprzystosowanie programu nauczania do poziomu ucznia, co wiąże się z bezpośrednim przeniesieniem obcojęzycznego wzorca. Mały Polak będzie czuł się zagubiony pośród natłoku nowych słów i wyrażeń (pamiętajmy, że nie będą mu się one kojarzyć natychmiast z zasłyszanymi, tak, jak małemu Anglikowi). Z drugiej strony niepotrzebna jest mu nauka kolejności liter w alfabecie, gdyż przypuszczalnie ma już to opanowanę.
- Niski stopień realizacji technicznej programu (chodzi tu głównie o powolność i niejednoznaczność reakcji na wciśnięcie klawisza, niemożliwość natychmiastowego przerwania testu itp.; reakcją dorosłego na takie kłopoty będzie namysł, ponowna próba lub przeczekanie, reakcją dziecka — panika).
- Monotonność przykładów i testów. (Tu jednak należy być ostrożnym w ocenie, gdyż może to być czynnikiem ułatwiającym obsługę programu przez nabranie rutyny).
- Polskie opisy i polecenia wydają się być nietrafnie dobrane i pojawianie się ich w angielskojęzycznym otoczeniu razi. Być może należało wprowadzić je w innej formie — oryginału polecenia w języku angielskim i następującego po nim tłumaczenia na język polski podanego w nawiasach. W programie zachowano występujące w wersji oryginalnej zabezpieczenie przed przerwaniem pracy za pomocą polecenia BREAK; po wydaniu go komputer zeruje się. Wydaje się, że należało usunąć taką formę zabezpieczenia, zastępując ją np. powrotem do głównego menu (obecnie realizowanego nietypowo poleceniem EDIT). Dziecko, które miało kontakt z komputerem ZX Spectrum, chcąc przerwać aktualnie wykonywany test odruchowo wciśnie kombinację klawiszy BREAK, niszcząc w ten sposób program.

Reasumując: wydaje się, że wybór tej serii programów nie był najlepszym z możliwych. Program jest przeznaczony dla indywidualnego użytkownika, i użytkowanie go w nauczaniu zbiorowym mija się z celem (myślę tu o wybranym na potrzeby polskiego szkolnictwa zgodnym z ZX Spectrum Elwro 800 Jr i jego pracy w sieci). Jeżeli zdecydowano się na niego, należało dokonać w nim gruntowniejszych (i bardziej przemyślanych) przeróbek, zarówno pod względem adaptacji sprzętowej (możliwość pracy w sieci, wersja dostarczana na dyskietkach), jak i merytorycznej (ukierunkowanie na polskiego odbiorcę). Być może jednak te przeróbki wymagałyby tak wielu zmian, że bardziej opłacałoby się napisać program od nowa...

Dariusz Adam Przygoda

KOMPUTEROWA SYMULACJA WIDM ATOMOWYCH

Jak wyglądają widma emisyjne różnych atomów? Czy można wyśledzić jakieś podobieństwa w widmach pierwiastków w ramach okresu lub grupy układu okresowego? Czy istnieje powiązanie między widmem a strukturą atomu? Te bardzo interesujące zagadnienia badano już od 1859 roku, kiedy G. R. Kirchoff i R. W. Bunsen zbudowali pierwszy spektroskop i zastosowali go do porównania widma słonecznego z widmami soli i metali. W 1915 roku N. Bohr i A. Sommerfeld stwierdzili związek między budową atomów i cząsteczek, a ich widmami. Uczyniło to spektroskopię jedną z głównych metod badania budowy atomów i cząsteczek i pozwoliło ustalić wiele z tego, co o strukturze materii możemy dzisiaj przeczytać w podręcznikach fizyki i chemii. Chyba więc warto zabawić się w „detektywów widm"?

W szkole na pewno dostępny jest prosty spektroskop pryzmatyczny. Zacznijmy od niego. Trzeba naocznie przekonać się jak „w naturze" wygląda np. widmo sodu. Niestety możliwości takiego przyrządu są bardzo ograniczone. Przede wszystkim jesteśmy w stanie obserwować widmo tylko w zakresie promieniowania widzialnego, a ponadto, ze względu na małą zdolność rozdzielczą spektroskopu, blisko położone linie będą się pokrywać. Warto by też jakoś zarejestrować obserwowane widma, jeżeli chcemy potem porównywać je między sobą. Nie pozostaje nam nic innego jak albo jakoś dostać się do laboratorium posiadającego odpowiednią aparaturę, albo — zabrać się za komputerową symulację widm.

Korzystając z wzoru Rydberga można, stosunkowo prosto, wygenerować widmo atomu wodoru uzyskując wyniki niewiele różniące się od obserwowanych w praktyce. Sprawa komplikuje się w przypadku cięższych pierwiastków. Na szczęście są obecnie dostępne katalogi widm, w których zebrano wyniki wieloletnich badań wielu uczonych.

Przedstawiony tu prosty program dla komputera ZX Spectrum korzysta właśnie z katalogowych wartości długości fal umieszczonych w liniach DATA. Umożliwia on jednoczesne przedstawienie na ekranie widm atomowych wodoru i litowców w zakresie od 80 do 4200 nm, czyli od dalekiego nadfioletu do średniej podczerwieni. Obserwowany zakres można zawęzić tak, żeby otrzymać najkorzystniejszy obraz. Widma wyglądają jak wąskie paski składające się z szeregu grubszych i cieńszych linii (rys. 1) i stąd pochodzi nazwa — widmo liniowe. Postać taka jest charakterystyczna dla wzbudzonych wolnych atomów lub jonów.

Pod widmami umieszczona jest informacja o ich zakresie oraz położeniu wskaźnika widocznego poniżej widma cezu w postaci małego krzyżyka. Wskaźnik można przesuwać w poziomie o 10 pikseli za pomocą klawiszy 5 i 8, a o 1 piksel wciskając jednocześnie CAPS SHIFT. Naciśnięcie klawiszy 6 lub 7 powoduje przestawienie wskaźnika pod inne widmo. Dokładność odczytu położenia wskaźnika zależy od zakresu widma. Zakres ten można zmieniać bądź przez naciśnięcie klawisza „z", a następnie podanie minimalnej i mak-




```
 5 REM ************************
 10 REM WIDMA ATOMOWE LITOWCÓW
 20 REM   A.M.GROSSMAN  1987
 25 REM ************************
 30 POKE 23609,20: POKE 23658,0
 40 REM * DEFINIOWANIE ZNAKOW *
 50 REM A="ą",B="ł",C="ż",O="ó"
 60 POKE 65480,8
 70 FOR i=0 TO 22: READ a: POKE
 (USR "A"+i),a: NEXT i
 80 DATA 0,16,56,16,0,56,0,0,0,
16,24,16,48,16,12,0,8,16,124,8,1
6,32,124
 90 DIM a$(2)
 100 LET ln10=LN 10: LET cz=.05:
 LET t=30
 105 REM ***********************
 110 REM * DEKLARACJA ZAKRESU *
 120 LET od=80: LET do=4200
 130 CLS
 140 PRINT AT 8,3;" WIDMA ATOMOW
E LITOWCÓW"
 150 PRINT AT 10,2;"W ZAKRESIE
 ";od;" - ";do;" nm"
 160 PAUSE 200
 170 LET min=od: LET max=do: LET
 y=38
 180 LET dl=max-min
 190 LET jed=dl/231
 200 LET mnoz=1/10^(INT (LN jed/
ln10))
 210 LET dokl=(INT (mnoz*jed+0.5
)/mnoz): LET d$=STR$ dokl: IF d$
(1)="." THEN  LET d$="0"+d$
 220 LET d$="  "+d$
 225 REM ***********************
 230 REM *   PLANSZA   *
 240 CLS : PLOT 0,38: DRAW 0,137
: PLOT 231,38: DRAW 0,137
 250 PRINT  PAPER 6;AT 18,22;"
     ";AT 18,0;"  Zakres :";min;
" - ";max;" nm"
 260 PRINT  PAPER 6;AT 19,0;"wsk
aźnik :"
 270 PRINT  PAPER 5;AT 20,0;" Zm
iana zakresu "; PAPER 7;" liczbo
wo :"; INVERSE 1;"z"
 280 PRINT AT 21,0;"wskaźnikiem
:"; INVERSE 1;"m"; INVERSE 0;" p
rzy min i max"
 290 GO SUB 300: GO TO 330
 300 PRINT #0;AT 0,0;"Dokładna d
ł. fali:"; INVERSE 1;"d"; INVERS
E 0;"  koniec :"; PAPER 7; INVER
SE 1;"q"
 310 PRINT #0;"ruch wskaźnika:";
 INVERSE 1;"5,6,7,8"; INVERSE 0;
"wolniej"; INVERSE 1;"+CS"
 320 RETURN
 325 REM ***********************
 330 REM * RYSOWANIE WIDMA *
 340 FOR n=0 TO 5
 350 RESTORE 1000+n*30
 360 READ a$: PRINT  BRIGHT 1;AT
 3*n,29;a$
 370 LET yf=165-n*24
 380 READ fala: IF fala>max [illegible]
 GO TO 380
 390 IF fala<min THEN  GO TO 420
 400 LET x=(fala-min)/jed
 410 PLOT x,yf: DRAW 0,10: GO TO
 380
 420 NEXT n
 430 BEEP cz,t
 435 REM ***********************
 440 REM *  WSKAZNIK *
 450 LET x=115: LET xs=x: LET ys
=y
 460 LET mm=0
 470 PRINT  PAPER 7; BRIGHT 1;AT
 19,21;d$; BRIGHT 0;" nm"
 480 LET wiecej=0
 490 LET fala=INT (mnoz*((x+0.5)
*jed+min))/mnoz
 500 LET f$=STR$ fala: LET l=LEN
 f$
```

symalnej wartości długości fali, bądź naciskając klawisz „m" przy dwóch położeniach wskaźnika ograniczających wymagany zakres widma. W obu przypadkach deklarowany zakres nie może być mniejszy niż 0,25 nm.

Ustawiając wskaźnik pod linią w widmie można, po naciśnięciu klawisza „d", uzyskać informację o dokładnej długości fali wskazywanej linii. Pojawienie się wykrzyknika sygnalizuje grupę kilku linii, a podawana wartość odnosi się do linii o najmniejszej długości fali. Zawężając zakres widma możemy rozdzielić tę grupę i obserwować pojedyncze linie.

Nasz komputerowy spektroskop dysponuje, jak widać, unikalną właściwością dowolnej zmiany zdolności rozdzielczej. W zależności od potrzeb można obserwować i porównywać widma w całości, lub badać szczegóły. W tym ostatnim przypadku możliwości naszego przyrządu są wręcz nieograniczone!

Niestety — tylko teoretycznie. W praktyce korzystamy z katalogu widm, a więc otrzymamy tylko takie informacje, jakie zostały zebrane za pomocą istniejących przyrządów. I to nie wszystkie. W rzeczywistym widmie linie różnią się między sobą jasnością (na zdjęciach fotograficznych — grubością), a ponadto obraz jednych może być wyraźny, innych nieco rozmyty. Są to wszystko dodatkowe dane umożliwiające wniknięcie w strukturę atomów, jednak, ze względu na ograniczone możliwości graficzne ZX Spectrum, trzeba było z nich zrezygnować. Przyjąłem najprostszą wersję uwzględniając tylko linie stosunkowo silne. Każda z nich, niezależnie od jasności, przedstawiona jest w postaci kreski o szerokości jednego piksela.

Dysponując lepszym komputerem można pokusić się o podzielenie linii na klasy wg podanej w katalogu jasności i każdą klasę przedstawić jako kreskę o innej grubości lub barwie.

Wybrałem widma atomowe litowców gdyż są one stosunkowo proste i stanowią wdzięczny materiał do porównań ze względu na występowanie charakterystycznych dubletów (dwóch blisko położonych linii), których odległości zwiększają się ze wzrostem liczby atomowej pierwiastka. Przedstawione dla porównania widmo wodoru reprezentowane jest przez stosunkowo dużą liczbę linii z serii Lymana i Balmera (w nadfiolecie i zakresie widzialnym) oraz kilka linii z serii Paschena i Bracketta (podczerwień).

Zainteresowanym mechanizmem powstawania widm polecam zabawę „Elektron w studni" („Bajtek" 4/87).

Działanie programu tłumaczą objaśnienia zawarte w listingu, stąd tylko kilka informacji dodatkowych:

Maksymalny zakres wyświetlanego widma deklarowany jest w linii 120 — zmienne „od—do". Jego przekroczenie powoduje komunikat o błędzie.

Zmienna „fala" przyjmuje albo dokładne (linie 380 i 890) albo przybliżone (linia 490) wartości długości fal.

Różnicę długości fal odpowiadającą jednemu pikselowi ekranu reprezentuje zmienna „jed", a jej wartość zaokrągloną przyjęto jako miarę dokładności wskazań wskaźnika — zmienna „dokl".

Ze względu na możliwość wystąpienia obrazu linii w sąsiednich pikselach przy różnicy długości fal większej niż wartość „jed", dla stwierdzenia obecności grupy linii widmowych (linie 840—910) badany jest zarówno obraz jak i dane z DATA w zakresie fala ±jed.

Program można łatwo dostosować do wyświetlania widm atomowych i jonowych innych pierwiastków zmieniając dane w liniach 1000—1160, ale zachowując układ: linia z symbolem pierwiastka, jedna linia z wartościami długości fal, linia z cyfrą 0 oznaczającą koniec danych.

```
 510 PRINT  BRIGHT 1;AT 19,10;"
          ";AT 19,15-1/2;f$+(" !
" AND wiecej>1)
 520 PLOT  INVERSE 1;xs,ys: DRAW
  INVERSE 1;0,5: PLOT  INVERSE 1
;xs+1,ys+4: PLOT  INVERSE 1;xs-1
,ys+4
 530 PLOT x,y: DRAW 0,5: PLOT x-
1,y+4: PLOT x+1,y+4
 540 LET xs=x: LET ys=y
 550 IF mm=1 THEN  PLOT xg,y: DR
AW 0,5
 560 IF INKEY$="" THEN  GO TO 56
0
 570 LET q$=INKEY$
 580 REM *PRZESUWANIE WSKAZNIKA*
 590 REM *   z=skok   *
 600 LET z=10
 610 IF CODE q$>9 THEN  GO TO 65
0
 620 IF CODE q$=8 THEN  LET q$="
5"
 630 IF CODE q$=9 THEN  LET q$="
8"
 640 LET z=1
 650 IF q$="5" AND x>z+3 THEN  L
ET x=x-z:
 660 IF q$="8" AND x<228-z THEN
 LET x=x+z
 670 IF q$="6" AND y>38 THEN  LE
T y=y-24
 680 IF q$="7" AND y<142 THEN  L
ET y=y+24
 685 REM **********************
 690 REM * ZMIANA ZAKRESU *
 700 REM * wskaznikiem *
 710 IF q$<>"m" THEN  GO TO 800
 720 BEEP cz,t
 730 IF NOT mm THEN  LET xg=x: L
ET nmin=fala: LET mm=1: GO TO 80
0
 740 LET max=fala
 750 IF nmin>max THEN  LET pom=n
min: LET nmin=max: LET max=pom
 760 IF max-nmin<.25 THEN  GO SU
B 950: GO TO 480
 770 IF nmin<od OR max>do THEN
GO SUB 920: GO TO 480
 780 LET min=nmin: GO TO 180
 790 [illegible] * liczbowo *
 [illegible] IF q$="z" THEN  BEEP cz,t:
INPUT "Zakres (nm):";nmin;" - ";
[illegible] GO TO 750
 810 IF q$="q" THEN  STOP
 820 IF q$<>"d" THEN  GO TO 470
 825 REM **********************
 830 [illegible] * DOKLADNA DL.FALI *
 840 IF NOT POINT (x,y+10) THEN
 GO SUB 960: GO TO 560
 850 LET wiecej=0: IF POINT (x+1
,y+10) OR POINT (x-1,y+10) THEN
 LET wiecej=1
 [illegible] LET odd=fala+jed: LET doo=f
ala-jed
 870 RESTORE 1160-((y-38)/24)*30
 [illegible] READ f: IF f>odd THEN  GO T
O [illegible]
 890 IF f>doo THEN  LET fala=f:
LET wiecej=wiecej+1: GO TO [illegible]
 900 BEEP cz,t: PRINT ;AT 19,21;
: FOR i=1 TO LEN d$: PRINT  BRIG
HT 1;" ";: NEXT i
 910 GO TO 500
 915 REM **********************
 920 [illegible] * REAKCJA NA BLAD *
 930 RANDOMIZE USR 3438: PRINT #
0;"Poza dopuszczalnym zakresem !
"
 940 GO TO 960
 950 RANDOMIZE USR 3438: PRINT #
0;"Zakres<0.25 [illegible] ";
 960 FOR i=1 TO 5: [illegible] cz,(i+5)
*3: NEXT i
 970 PAUSE 50: GO SUB 300
 980 RETURN
 985 [illegible] **********************
 990 REM *DLUGOSCI FAL**
1000 DATA "H"
1010 DATA 4051.14,2625.13,1875.1
1,1281.81,1093.81,656.285,656.27
3,486.133,410.174,397.007,388.90
5,383.539,379.79,377.063,375.015
,373.437,121.567,102.572,97.253,
94.974,93.78,93.075,92.623,92.31
5,92.096,91.935,91.813,91.718,91
.643,91.582
1020 DATA 0
1030 DATA "Li"
1040 DATA 812.645,812.623,670.79
1,670.776,610.365,610.354,497.17
5,497.166,460.289,460.283,427.31
3,427.307,413.262,413.256,398.55
4,398.548,391.535,391.530,323.26
6,274.12,256.231,247.506,242.543
1050 DATA [illegible]
1060 DATA "Na"
1070 DATA 1140.378,1138.145,1083
.487,1074.644,819.482,818.326,58
9.592,588.995,568.822,568.263,45
4.519,454.163,449.765,449.418,43
9.334,439.003,330.298,330.237,28
5.301,285.281,268.043,268.034
1080 DATA 0
1090 DATA "K"
1100 DATA 1177.283,1176.962,1169
.021,1102.267,1048.711,1047.963,
959.783,959.57,890.402,890.219,8
50.511,850.345,825.174,825.018,7
69.896,766.49,696.467,693.628,58
3.189,581.215,535.957,534.297,53
3.969,532.328,511.225,509.92,509
.717,508.423,496.503,495.615,495
.081,494.201,486.976,486.348,485
.609,484.986,464.237,464.188,404
.721,404.414,344.738,344.643,310
.204,310.179,303.492,299.222,296
.324
1110 DATA [illegible]
1120 DATA "Rb"
1130 DATA 2731.35,2293.115,2252.
88,1366.505,1323.527,1007.61,100
5.52,952.34,886.97,794.76,792.52
6,780.023,775.765,629.923,615.96
2,607.075,565.374,557.878,543.15
3,536.26,421.556,420.185,359.159
,358.708,335.089,334.872,322.798
,315.753,311.305,311.257
1140 DATA [illegible]
1150 DATA "Cs"
1160 DATA 3939.8,3612.8,3490,309
5.2,3010.2,2424.8,2334,2303.2,14
69.493,1358.831,920.854,917.233,
894.359,894.335,876.141,852.124,
852.103,807.902,801.572,799.068,
794.388,727.996,722.853,697.329,
687.045,682.465,672.328,662.865,
658.602,647.263,643.197,621.31,6
01.049,584.514,459.32,455.531,38
8.863,388.858,387.617,387.612,36
1.741,361.145,348.006,347.681,33
4.749,334.744
1170 DATA [illegible]
9999 CLEAR : SAVE "WIDMA H-Cs" L
INE 10
```

ANDRZEJ M. GROSSMAN

SEMINARIUM „INFORMIKA”

ASEMBLER — GENS 3

Część 9

Ostatni odcinek kursu asemblera przeznaczamy zgodnie z zapowiedzią na dokończenie opisu programu „Mini-monitora”. W dalszym ciągu wydruków przedstawione zostaną realizacje pozostałych zleceń monitora wraz z niezbędnymi dla ich realizacji podprogramami.

Jeżeli czytelnik zamierza uruchomić program w całości, przypominam, że w I części programu występuje zmienna KOMPL, której należy, po dopisaniu aktualnie publikowanych wydruków, nadać wartość różną od zera. Wtedy dopiero zostaną asemblowane zlecenia, których realizacje nie występowały w poprzedniej części. Podobnie jak poprzednio wydruk został podzielony na kilka funkcjonalnie odrębnych części, które zostaną kolejno dość szczegółowo omówione.

Wykorzystane zmienne i procedury zewnętrzne

SABYT1 — procedura zapisuje na taśmie magnetofonowej blok danych. Właściwa procedura w ROM, rozpoczyna się o 4 bajty wcześniej (SABYTE #04C2), zostaje wtedy zapisany na stos, adres powrotu z procedury w przypadku naciśnięcia klawisza BREAK. Ponieważ jest on umieszczony w ROM wchodzimy do procedury dalej. Przypominam, że w głównej pętli programowej przed wykonaniem jakiegokolwiek zlecenia najpierw umieszczony zostaje na stosie adres powrotu do programu monitora, co gwarantuje w przypadku przerwania zlecenia powrót do głównej pętli. Ta sama uwaga odnosi się do następnej procedury.

LDSTAR — procedura odczytuje blok danych z taśmy magnetofonowej. Standardowo wywołuje się procedurę LDBYTE — pod adresem #0556.

DECEP — procedura zamienia liczbę zapisaną w znakach ASCII, na format zmiennoprzecinkowy (5 bajtów) i umieszcza ją na stosie kalkulatora.

NUMERI — procedura sprawdza czy kod ASCII (podany do akumulatora) odpowiada cyfrze od zera do dziewięciu. Jeśli nie to CY = 1.

EPTOBC — procedura zdejmuje ze stosu kalkulatora liczbę całkowitą (dwubajtową) i wpisuje ją do pary rejestrów BC.

HLHLDE — procedura mnoży zawartość pary DE przez parę HL, wynik w HL. Jeśli wystąpi nadmiar wskaźnik CY = 1.

CHADD — adres znaku dla interpretacji, zmienna systemowa używana przez procedury interpretera BASIC-a.

Procedury wyprowadzania na ekran

PRCHRS — procedura wyprowadza na ekran każdy kod większy lub równy kodowi spacji. Znaki kontrolne mające kody mniejsze od kodu spacji będą zastępowane znakiem kropki.

PRCODE — jw., z tym że najstarszy bit kodu zostaje wyzerowany. Jest to przydatne, gdyż przeważnie w ROM ostatni znak wyprowadzanego tekstu jest sygnalizowany ustawieniem tego bitu, co utrudniłoby przy odczycie zawartości pamięci poprawną interpretację programu.

PR1PEK — wyprowadza dziesiętnie zawartość jednej komórki o adresie podanym w HL.

PR2PEK — wyprowadza dziesiętnie zawartość dwóch komórek o adresie podanym w HL. Jest ona traktowana jako liczba dwubajtowa.

```
3540 ;---------------------------
3550 ;MINI-MONITOR CZ.2
3560 ;Kurs asemblera Gens3 cz.9
3570 ;(C) Tadeusz Basista 1988
3580 ;---------------------------
3590 ENTER   EQU   #0D
3600 SABYT1  EQU   #04C6
3610 LDSTAR  EQU   #056C
3620 DECFP   EQU   #2C9B
3630 NUMERI  EQU   #2D1B
3640 FPTOBC  EQU   #2DA2
3650 HLHLDE  EQU   #30A9
3660 CHADD   EQU   23645
3670 ;---------------------------
3680 ;wypr.znaku ASCII
3690 PRCHR$ CP    " "
3700        JR    NC,PRCHRA
3710        LD    A,"."
3720 PRCHRA RST   PRINTA
3730        LD    A," "
3740        RST   PRINTA
3750        RET
3760
3770 PRCODE LD    A,(HL)
3780        RES   7,A
3790        JR    PRCHR$
3800 ;---------------------------
3810 ;wypr.dzies.1 komorki
3820 PR1PEK LD    C,(HL)
3830        LD    B,0
3840        CALL  PR2DEC
3850        RET
3860 ;---------------------------
3870 ;wypr.dzies.2 komorek
3880 PR2PEK LD    C,(HL)
3890        INC   HL
3900        LD    B,(HL)
3910        CALL  PR2DEC
3920        RET
3930 ;---------------------------
3940 ;wypr.binarnie rejestru C
3950 PR1BIN LD    B,8
3960 PR1BI1 LD    A,"0"
3970        RL    C
3980        JR    NC,PR1BI2
3990        INC   A
4000 PR1BI2 PUSH  BC
4010        RST   PRINTA
4020        POP   BC
4030        DJNZ  PR1BI1
4040        RET
4050 ;---------------------------
4060 ;wypr.binarnie rej. HL
4070 PR2BIN PUSH  HL
4080        LD    C,H
4090        CALL  PR1BIN
4100        LD    A," "
4110        RST   PRINTA
4120        POP   HL
4130        LD    C,L
4140        CALL  PR1BIN
4150        RET
4160 ;---------------------------
4170 ;wypr.dzies.rej.HL
4180 PDECHL PUSH  HL
4190        POP   BC
4200        CALL  PR2DEC
4210        RET
4220 ;---------------------------
4230 ;wypr.szesn.rej.HL
4240 PHEXHL PUSH  HL
4250        POP   BC
4260        CALL  PR2HEX
4270        RET
4280 ;---------------------------
```

PR1BIN — wyprowadza binarnie zawartość rejestru C jako ośmioznakowy ciąg zer lub jedynek. Nieznaczące zera nie są pomijane.

PR2BIN — wyprowadza binarnie zawartość pary HL. Najpierw starszy bajt, następnie jedna spacja i młodszy bajt.

PDECHL — wyprowadza dziesiętnie zawartość pary HL.

PHEXHL — wyprowadza szesnastkowo zawartość pary HL.

Obsługa bufora klawiatury

TESHEX — procedura sprawdza czy kod znajdujący się w akumulatorze jest kodem litery A,B,C,D,E,F. Jeśli nie to CY = 1.

TESBIN — procedura sprawdza czy kod znajdujący się w akumulatorze jest kodem cyfry 0,1. Jeśli nie to CY = 1.

Wymienione procedury oraz wcześniej omówiona procedura NUMERI znajdująca się w ROM, są wykorzystywane w trakcie wprowadzania parametrów. Dzięki temu, klawiatura po prostu nie reaguje na znaki niedopuszczalne dla danego typu parametru (decymalnego, szesnastkowego, binarnego). Oczywiście dla trybu ASCII kontrola ta nie jest prowadzona.

INPUT1 — procedura wprowadza jeden parametr w dowolnym trybie do bufora. Na wstę-

pie wywoływana jest procedura odczytu klawiatura KLAW1.

Naciśnięcie klawisza DELETE (CY = 1) powoduje przerwanie procedury. Następnie sprawdza się czy nie nastąpiło naciśnięcie klawisza ENTER lub „przecinek". W obu przypadkach wprowadzanie parametru jest zakończone (skok do etykiety INPUT6). Jeśli nie, to sprawdza się czy naciśnięty klawisz jest jednym z klawiszy zmiany trybu. Jeśli tak, to oczekujemy na inny, ponieważ te znaki nie będą wprowadzane do bufora. Następuje teraz sekwencja kontrolnych instrukcji, którą łatwiej prześledzić, jeśli zobrazujemy binarnie kody znaków określających tryb. Trzy najmniej znaczące bity tych kodów mają następujące wartości:

\# — 011 tryb szesnastkowy
$ — 100 tryb ASCII
% — 101 tryb binarny
& — 110 tryb dziesiętny.

Stąd widać, że pytanie o bit drugi identyfikuje tryb szesnastkowy, następne pytanie o bit pierwszy identyfikuje tryb dziesiętny, i ostatnie pytanie o bit zerowy identyfikuje tryb binarny. Po ustaleniu trybu, następuje kontrola (poza trybem ASCII) zgodności i jeśli jest poprawna kod zostaje wprowadzony do bufora, a znak odpowiadający kodowi wyprowadzany na ekran (fragment od etykiety INPUT5). Dodatkowo, jeśli wprowadzono kod przecinka, zostaje wywołana procedura KLAW3 wyprowadzająca znak aktualnego trybu parametrów.

```
4290 ;czy Kod od "A" do "G"
4300 TESHEX CP   "A"
4310        RET  C
4320        CP   "G"
4330        CCF
4340        RET
4350 ;------------------------
4360 ;czy Kod "0" lub "1"
4370 TESBIN CP   "0"
4380        RET  C
4390        CP   "2"
4400        CCF
4410        RET
4420 ;------------------------
4430 ;wprowadzenie Parametru.
4440 INPUT1 LD   HL,BUFOR+1
4450 INPUT2 CALL KLAW1
4460        RET  C
4470        CP   ","
4480        JR   Z,INPUT6
4490        CP   ENTER
4500        JR   Z,INPUT6
4510
4520        CALL TESTTR
4530        JR   NC,INPUT2
4540
4550        BIT  2,(IY+71)
4560        JR   NZ,INPUT4
4570 ;tryb nie szesnastk.
4580
4590        CALL TESHEX
4600        JR   C,INPUT3
4610        JR   INPUT5
4620
4630 INPUT3 CALL NUMERI
4640        JR   C,INPUT2
4650        JR   INPUT5
4660
4670 INPUT4 BIT  1,(IY+71)
4680        JR   NZ,INPUT3
4690 ;tryb dzies.
4700
4710        BIT  0,(IY+71)
4720        JR   Z,INPUT5
4730 ;tryb ASCII
4740
4750        CALL TESBIN
4760        JR   C,INPUT2
4770
4780 INPUT5 LD   (HL),A
4790        INC  HL
4800        PUSH HL
4810        RST  PRINTA
4820        POP  HL
4830        JR   INPUT2
4840
4850 INPUT6 LD   (HL),ENTER
4860        PUSH AF
```



```
4870        LD   A,","
4880        RST  PRINTA
4890        POP  AF
4900        CP   ","
4910        CALL Z,KLAW3
4920        AND  A
4930        RET
4940 ;------------------------
```

Interpretacja parametrów

PARDEC — procedura zamienia ciąg znaków ASCII w buforze na liczbę całkowitą dwubajtową umieszczoną w rejestrze BC. Wykorzystano w niej znacznie ogólniejszą procedurę DECEP znajdującą się w ROM-ie. Procedura ta zamienia każdą liczbę (znajdującą się w zakresie) zmiennoprzecinkową będącą ciągiem znaków ASCII. Interpretuje odpowiednio również część całkowitą, część ułamkową, kropkę dziesiętną, znak wykładnika u mantysy, jeśli podano liczbę w zapisie naukowym. Wymagane jest wpisanie do zmiennej systemowej CHADD adresu pierwszego znaku liczby i wpisanie do akumulatora kodu tego znaku. Jeśli poprzedzimy ten ciąg znaków kodem funkcji BIN, to można zinterpretować za pomocą tej procedury ciąg znaków przedstawiający liczbę w postaci binarnej.

PARBIN — procedura zamienia ciąg znaków ASCII w buforze na liczbę całkowitą dwubajtową umieszczaną w rejestrze BC. W ciągu tym mogą wystąpić wyłącznie kody zera i jedynki.

PARHEX — procedura zmienia ciąg znaków ASCII w buforze na liczbę dwubajtową umieszczoną w rejestrze BC. W ciągu tym mogą wystąpić jedynie kody cyfr od zera do dziewięciu i liter od A do F. Procedura jest obszerniejsza, gdyż w ROM brak odpowiedniej procedury interpretującej liczby w postaci szesnastkowej. W procedurze tej w rejestrze HL przechowywany jest adres aktualnie interpretowanego znaku, a w rejestrze BC aktualny wynik. Na wstępie BC jest zerowane, a HL ustawiane na adres poprzedzający pierwszy znak. Następnie pobierany jest do akumulatora kolejny znak i sprawdzone zostaje czy nie jest to kod ENTER. Jeśli tak, procedura kończy się. W przeciwnym wypadku zapamiętywany jest na stosie rejestr HL, następnie zawartość rejestrów BC jest wpisywana do HL. Ze względu na brak rozkazu zamiany BC z HL najprościej wykonać można taką operację używając stosu. Teraz zawartość HL jest mnożona przez 16. Wykorzystana została tu procedura HLHLDE z ROM. Wynik mnożenia jest w HL. Następuje przepisanie wyniku do BC i odtworzenie pierwotnej zawartości HL (tzn. adresu znaku). Jeśli w wyniku mnożenia został przekroczony zakres to nastąpi zakończenie procedury z ustawionym wskaźnikiem CY. Jeśli wynik jest poprawny kod znaku jest pobierany do akumulatora i pomniejszany o 48, jeśli jest kodem cyfry i dodatkowo pomniejszany o 7, jeśli jest kodem litery. Teraz od etykiety PARHE2 liczba powstała z aktualnie zamienionego kodu jest dodawana do aktualnego wyniku.

PARASC — procedura wpisuje bezpośrednio kod ASCII z bufora do rejestru C.

```
4950 ;zamiana Param.dziesietn.
4960 ;wynik w BC.Jesli >65535
4970 ;to CY=0
4980 PARDEC LD   HL,BUFOR+1
4990 PARDE1 LD   (CHADD),HL
5000        LD   A,(HL)
5010        CALL DECFP
5020        CALL FPTOBC
5030        RET
5040 ;-------------------------
5050 ;zamiana Param.bin.  jw
5060 PARBIN LD   HL,BUFOR
5070        LD   A,#C4
5080        LD   (HL),A
5090        CALL PARDE1
5100        RET
5110 ;-------------------------
5120 ;zamiana Param.szesnast.
5130 PARHEX LD   BC,0
5140        LD   HL,BUFOR
5150 PARHE1 INC  HL
5160        LD   A,(HL)
5170        CP   ENTER
5180        RET  Z
5190
5200        PUSH HL
5210        PUSH BC
5220        POP  HL
5230        LD   DE,16
5240        CALL HLHLDE
5250        PUSH HL
5260        POP  BC
5270        POP  HL
5280        RET  C
5290
5300        LD   A,(HL)
5310        SUB  #30
5320        CP   10
5330        JR   C,PARHE2
5340
5350        SUB  7
5360 PARHE2 PUSH HL
5370        PUSH BC
5380        POP  HL
5390        LD   D,0
5400        LD   E,A
5410        ADD  HL,DE
5420        PUSH HL
5430        POP  BC
5440        POP  HL
5450        JR   PARHE1
5460 ;-------------------------
5470 ;zamiana Param. ASCII
5480 PARASC LD   HL,BUFOR+1
5490        LD   B,0
5500        LD   C,(HI
5510        RET
5520 ;-------------------------
```

USTIPA — procedura realizuje pobranie i odpowiednią do typu parametru interpretację. Wywoływana jest na wstępie procedura INPUT1, a następnie realizowane w zależności od aktualnego trybu wywołanie odpowiedniej procedury interpretującej. Powtarza się tu ten sam schemat rozróżniania typów parametru, który wystąpił w procedurze INPUT1. Rozdzielenie procesu kontroli od interpretacji wydaje się jednak celowe, gdyż łatwo sobie wyobrazić jak nieczytelny byłby algorytm w przypadku ich połączenia.

USTPAR — procedura pobiera i interpretuje od 1 do 3 parametrów. Ilość parametrów należy podać do rejestru E. Do rejestru HL podać adres pierwszej komórki pamięci przeznaczonej na przechowywanie parametrów. Oczywiście ilość parametrów może być większa, jeśli zarezerwujemy odpowiedni obszar pamięci. Zauważmy, że omawiane procedury wejścia realizują kolejno coraz ogólniejsze zadania, tak że w głównej pętli programowej wywoływana jest jedynie raz stojąca najwyżej w hierarchii procedura USTPAR. Efekt ten nie jest przypadkowy i jest celem, do którego powinniśmy zmierzać starając się programować strukturalnie. Przybliżonym miernikiem tego, w jakim stopniu nam się to udało może być liczba instrukcji skoku w programie. W omawianym programie nie przekracza ona 5% ogólnej liczby instrukcji, co autor uważa za wynik zupełnie przyzwoity. Ostatnią procedurą, którą omówimy przed opisem realizacji zleceń jest procedura listowania zawartości pamięci.

LIST1 — liczba wyprowadzanych jednorazowo komórek pamięci jest przechowywana w rejestrze E, adres, od którego zawartość pamięci będzie wyprowadzana jest wpisywany do rejestru BC. Następuje teraz kolejne wyprowadzenie tego adresu najpierw w postaci dziesiętnej, następnie w postaci szesnastkowej (procedury PR2DEC i PRSHEX). Dalej jest wyprowadzana zawartość komórki pamięci w postaci szesnastkowej i w postaci ASCII ze zgaszonym 7 bitem (procedury PR1HEX i PRCODE). Na końcu wyprowadzana jest dziesiętnie zawartość komórki i zawartość dwóch kolejnych komórek traktowana jako liczba dwubajtowa (procedury PR1PEK i PR2PEK). Teraz zawartość ekranu jest przesuwana o jedną linię do góry (procedura SCROLL) i następuje wyprowadzenie kolejnej linii.

```
5530 ;wprowadz.  jednego Param.
5540 UST1PA CALL INPUT1
5550        RET  C
5560        BIT  2,(IY+71)
5570        JR   Z,USTHEX
5580        BIT  1,(IY+71)
5590        JR   NZ,USTDEC
5600        BIT  0,(IY+71)
5610        JR   NZ,USTBIN
5620        CALL PARASC
5630        RET
5640
5650 USTHEX CALL PARHEX
5660        RET
5670
5680 USTDEC CALL PARDEC
5690        RET
5700
5710 USTBIN CALL PARBIN
5720        RET
5730 ;-------------------------
5740 ;wprowadz. 1,2 lub 3
5750 ;Parametrow.Ilosc Par.w E
5760 USTPAR LD   HL,PARAM1
5770 USTPA1 PUSH DE
5780        PUSH HL
5790        CALL UST1PA
5800        POP  HL
5810        POP  DE
5820        RET  C
5830
5840        LD   (HL),C
5850        INC  HL
5860        LD   (HL),B
5870        INC  HL
5880        DEC  E
5890        JR   NZ,USTPA1
5900        RET
5910 ;-------------------------
5920 ;wydruk zawart. Pamieci
5930 ;liczba Komorek w rej.E
5940 LIST1  PUSH DE
5950        CALL PRTAT
5960        DEFB 32
5970        LD   BC,(PARAM1)
5980        PUSH BC
5990        CALL PR2DEC
6000        CALL PRTAT
6010        DEFB 26
6020        POP  BC
6030        PUSH BC
6040        CALL PR2HEX
6050        POP  HL
6060        PUSH HL
6070        LD   A,(HL)
6080        CALL PR1HEX
6090        LD   A," "
6100        RST  PRINTA
6110        POP  HL
6120        PUSH HL
6130        CALL PRCODE
6140        POP  HL
6150        PUSH HL
6160        CALL PR1PEK
6170        CALL PRTAT
6180        DEFB 12
6190        POP  HL
6200        PUSH HL
6210        CALL PR2PEK
6220        POP  HL
6230        INC  HL
6240        LD   (PARAM1),HL
6250
6260        CALL SCROLL
6270        POP  DE
6280        DEC  E
6290        JR   NZ,LIST1
6300        RET
6310 -------------------------
```

Realizacja zleceń monitora

Przypomnijmy, że w głównej pętli programowej wprowadzono parametry, o ile wystąpiły, to zostały dodatkowo zapamiętane w dwóch kompletach rejestrów kolejno w HL, DE, BC i drugi raz w IX, BC, DE.

STALA — procedura zapełnia obszar pamięci od adresu w HL (pierwszy parametr), o długości w BC (drugi parametr), wartością będącą trzecim parametrem. Użyto w tym celu typowego „chwytu" programowego dla procesora Z-80. Daną wartość wpisuje się do komórki o początkowym adresie, a następnie dokonuje się przesłania bloku (o długości o jeden mniejszej niż długość zadanego obszaru) pod adres większy o jeden od adresu początkowego.

BLOKI — procedura kopiuje blok pamięci od adresu w HL (pierwszy parametr), pod adres docelowy w DE (drugi parametr), o długości bloku w BC (trzeci parametr). Kopiowanie jest „inteligentne", tzn. sprawdza się czy adres docelowy jest mniejszy od źródłowego, czy też większy. W pierwszym przypadku zostaje wykonana instrukcja LDIR (zawartość pamięci jest kopiowana od najniższego adresu). W drugim przypadku

zostaje wykonana instrukcja LDDR (zawartość pamięci jest kopiowana od najwyższego adresu w kierunku niższych adresów). Takie postępowanie jest konieczne, jeśli chcemy poprawnie kopiować w przypadku, gdy obszar źródłowy i docelowy „zachodzą na siebie".

DODAJ, ODEJM, WYNIK — procedury nie wymagają specjalnego komentarza, warto jedynie zauważyć, że w przypadku wykrycia nadmiaru następuje powrót z procedur bez wyprowadzenia wyniku. O ile wynik jest poprawny, to zostaje wyprowadzony od razu w postaci szesnastkowej i dziesiętnej.

ZAMIEN — procedura wymaga jednego parametru. Ze względu na sześć możliwych kombinacji zamiany parametrów, przyjęto ze względów praktycznych kompleksowe rozwiązanie problemu. Wprowadzony parametr (dowolnego typu) jest od razu zamieniany na wszystkie cztery możliwe typy. Kolejno na postać szesnastkową, dziesiętną, ASCII i od nowej linii w postaci binarnej.

```
6320 STALA   EXX
6330         LD   (HL),C
6340         PUSH DE
6350         POP  BC
6360         DEC  BC
6370         PUSH HL
6380         POP  DE
6390         INC  DE
6400         LDIR
6410         EXX
6420         RET
6430 ;-----------------------
6440 BLOKI   EXX
6450         AND  A
6460         PUSH HL
6470         SBC  HL,DE
6480         POP  HL
6490         JR   C,BLOK1
6500         LDIR
6510         DEC  DE
6520         PUSH DE
6530         JR   BLOK2
6540
6550 BLOK1   ADD  HL,BC
6560         DEC  HL
6570         PUSH HL
6580         EX   DE,HL
6590         ADD  HL,BC
6600         EX   DE,HL
6610         DEC  DE
6620         POP  HL
6630         PUSH DE
6640         LDDR
6650 BLOK2   POP  HL
6660         EXX
6670         RET
6680 ;-----------------------
6690 DODAJ   EXX
6700         ADC  HL,DE
6710         JR   WYNIK
6720
6730 ODEJM   EXX
6740         SBC  HL,DE
6750
6760 WYNIK   EXX
6770         RET  C
6780
6790         EXX
6800         PUSH HL
6810         CALL PHEXHL
6820         LD   A," "
6830         RST  PRINTA
6840         POP  HL
6850         CALL PDECHL
6860         EXX
6870         RET
6880 ;-----------------------
6890 ZAMIEN  EXX
6900         PUSH HL
6910         CALL PHEXHL
6920         POP  HL
6930         PUSH HL
6940         LD   A," "
6950         RST  PRINTA
6960         POP  HL
6970         PUSH HL
6980         CALL PDECHL
6990         LD   A," "
7000         RST  PRINTA
7010         POP  HL
7020         PUSH HL
7030         LD   A,L
7040         CALL PRCHR$
7050         LD   A," "
7060         RST  PRINTA
7070         POP  HL
7080         PUSH HL
7090         CALL SCROLL
7100         CALL PRTAT
7110         DEFB 32
7120         POP  HL
7130         CALL PR2BIN
7140         EXX
7150         RET
7160 ;-----------------------
```

Zanim przejdziemy do opisu zleceń odczytu i zapisu na taśmie magnetofonowej przypomnijmy kilka podstawowych faktów. Procedury zapisu i odczytu umieszczone w ROM rozróżniają dwa typy bloków:

Blok danych: (o dowolnej długości) i

Blok nagłówka (poprzedza on blok właściwych danych) składający się z 17 bajtów mających następujące znaczenie:

Tabela 1

NR	Zawartość
1	Typ danych: 0 — program w BASIC-u, 1 — tablica liczbowa, 2 — tablica tekstowa, 3 — kod binarny
2—11	Nazwa danych (dziesięć znaków ASCII)
12—13	Mniej znaczący i bardziej znaczący bajt adresu początkowego ładowania
14—15	Dla programów w BASIC-u numer linii, od której program uruchamia się automatycznie
16—17	Długość bloku danych (mniej znaczący i bardziej znaczący bajt)

Fizycznie na taśmie dodatkowo są zapisywane jeden bajt przed blokiem równy 0 dla bloku nagłówka i 255 dla bloku danych. oraz po bloku jeden bajt kontroli parzystości. Standardowe procedury zapisu i odczytu #04C2 oraz #0556 wymagają następujących parametrów:

Tabela 2

Rejestr	SAVE	LOAD	VERIFY
IX	Adres początku	— „ —	— „ —
DE	Długość	— „ —	— „ —
A	0 — dla nagłówka 255 — dla danych		
CY		ustawiony	wyzerowany

Widać stąd, że można zapisać blok typu nagłówka o długości większej niż 17 bajtów. Np. popularny program COPY-COPY nie potrafi skopiować takich bloków. Zrealizowana w monitorze procedura ODCZYT wczytuje dowolny blok. Jedynym parametrem wymaganym jest adres w pamięci, od którego począwszy blok ma być ładowany. Jeśli odczyt przebiegł prawidłowo jest wyprowadzany na ekran adres ostatniego wczytanego bajtu. Jakiego typu blok został wczytany możemy dowiedzieć się sprawdzając zawartość komórki pamięci o adresie o jeden mniejszym od adresu ładowania.

ZAPIS — procedura wymaga trzech parametrów: adresu początkowego, długości bloku, i typu bloku. W ostatnim parametrze należy podać 0, jeśli chcemy zapisać blok nagłówka, lub 255, jeśli chcemy zapisać blok danych.

PAMIEC — procedura wyprowadza zawartość 16 kolejnych komórek pamięci wraz z adresami w formacie opisanym w procedurze LIST1. Jedynym argumentem jest adres początkowy. Naciśnięcie klawisza DELETE powoduje powrót do głównej pętli programowej. Modyfikacja zawartości pamięci jest możliwa po naciśnięciu klawisza „A". W tym podzleceniu po każdym wpisie automatycznie jest zwiększany aktualny adres i nowa zawartość pamięci listowana w pełnej postaci.

```
7170 ODCZYT  DEC  IX
7180         LD   DE,#FFFF
7190         LD   C,#0E
7200         SCF
7210         EX   AF,AF'
7220         DI
7230         CALL LDSTAR
7240         EI
7250         RET  C
7260         LD   HL,#FFFD
7270         SBC  HL,DE
7280         CALL PDECHL
7290         RET
7300 ;-----------------------
7310 ZAPIS   LD   A,(PARAM3)
7320         CALL SABYT1
7330         EI
7340         RET  NC
7350
7360         PUSH IX
7370         POP  HL
7380         DEC  HL
7390         DEC  HL
7400         CALL PDECHL
7410         RET
7420 ;-----------------------
7430 PAMIEC  LD   (PARAM2),BC
7440         LD   E,16
7450         CALL LIST1
7460         CALL SCROLL
7470         CALL KLAW1
7480         RET  C
7490
7500         CP   "A"
7510         JR   Z,WPIS
7520         JR   PAMIEC
7530
7540 WPIS    CALL PRTAT
7550         DEFB 32
7560         LD   HL,(PARAM2)
```

```
7570         LD   (PARAM1),HL
7580         PUSH HL
7590         CALL PDECHL
7600         CALL PRTAT
7610         DEFB 26
7620         POP  HL
7630         CALL PHEXHL
7640         LD   A,":"
7650         RST  PRINTA
7660         CALL UST1PA
7670         RET  C
7680
7690         LD   HL,(PARAM2)
7700         LD   (HL),C
7710         LD   B,1
7720         CALL ERASE
7730         LD   E,1
7740         CALL LIST1
7750         LD   HL,(PARAM2)
7760         INC  HL
7770         LD   (PARAM2),HL
7780         JR   WPIS
7790 ;-------------------------
7800 SKOK    LD   HL,MONIT
7810         PUSH HL
```

```
7820         PUSH BC
7830         POP  HL
7840         JP   (HL)
7850
7860 MONIT   EI
7870         RET
7880 ------------------------
```

Uwagi końcowe

Przedstawiony powyżej program zajmuje 1128 bajtów. Wymagany obszar Tablicy Symboli wynosi 1394 bajty, a cały program źródłowy liczy 10 555 bajtów. Ten nieco nużący opis (i przydługawy) konkretnego programu wydaje się jednak konieczny, jeśli czytelnik zamierza naprawdę zrozumieć o co właściwie chodzi. Oczywiście jedynym prawdziwym sprawdzianem umiejętności będzie napisanie własnego programu. Jednakże w przypadku asemblera, opanowanie jedynie listy rozkazowej procesora jest drogą prowadzącą donikąd. Stąd autorowi wydało się celowe przedstawienie większego przykładu ilustrującego typowe techniki programowania. Jeżeli uważny czytelnik dokona modyfikacji programu i po pewnym czasie niewiele zostanie z oryginału autor będzie w pełni usatysfakcjonowany. Praktyka bowiem pokazuje, że w przypadku języka maszynowego najtrudniej zacząć pisać... a jeszcze trudniej pisać tak, żeby po jakimś czasie zrozumieć siebie samego.

TADEUSZ BASISTA

MONITOR KOMPUTEROWY

W ostatnich latach pojawiają się coraz nowocześniejsze rozwiązania ekranów służących do wyświetlania informacji, od wyświetlaczy ciekłokrystalicznych aż po kolorowe ekrany plazmowe. Jeżeli jednak racjonalnie przyjrzymy się statystyce występowania poszczególnych rozwiązań to musimy przyznać, że zdecydowane pierwszeństwo przypada tutaj poczciwej staruszce-lampie kineskopowej. Dowcipni twierdzą, że jest to jedyna lampa, która opiera się jak na razie erze mikroprocesorów. Mimo że lampa ta posiada już sędziwy wiek, potrafiła tak zmienić swe oblicze, że jej pozycja nadal wydaje się niezagrożona.

Weźmy dla przykładu zdawałoby się tak prosty jej element jak szyba ekranu. Z uwagi na obciążenia jakim podlega, zgniatana siłą co najmniej kilkuset kilogramów przez ciśnienie otaczającego powietrza, oraz z uwagi na minimalizację zniekształceń geometrii obrazu wewnętrzna jej powierzchnia musi mieć kształt zbliżony do sferycznego. Mimo tego można uzyskać idealnie płaską powierzchnię ekranu. Niestety nie wystarczy tutaj wykonanie ekranu w postaci soczewki płasko-wypukłej, gdyż dałoby to zupełnie przeciwny efekt, dodając nowe zniekształcenia geometrii oglądanego obrazu. Aby uzyskać płaski ekran, powierzchnię czołową ekranu wykonuje się spiekając ze sobą miliony włókien światłowodowych. Tak przygotowany blok tnie się na tafle, a następnie nadaje im odpowiedni kształt. Taka szyba ma dziwne właściwości. Oglądana wydaje się być matowa, jednak gdy przyłożymy do jej wklęsłej strony dokładnie jakiś obraz, to zostanie on przeniesiony na jej drugą stronę w ten sposób, że obserwatorowi wydaje się, iż jest on położony na jej płaskiej powierzchni. Taka szyba pozwala nie tylko uczynić ekran kineskopu płaskim, ale także całkowicie eliminuje błąd paralaksy powstający np. przy próbie zmierzenia odcinka na ekranie „normalnego" kineskopu. Dodatkowo poprawia się znacznie kontrast oglądanych obrazów, gdyż nie występuje efekt podświetlania luminoforu poprzez światło odbite od powierzchni czołowej szyby przedniej. Mimo że opis idei tego rozwiązania wydaje się prosty, trudności technologiczne napotykane przy realizacji tego pomysłu są tak znaczne, że jak dotąd stosuje się je tylko w złożonych monitorach o bardzo dużej rozdzielczości.

Innym rozwiązaniem wprowadzanym w ostatnich latach jest dynamiczna korekcja ogniskowania strumienia elektronów. Jeżeli przyjrzymy się ekranowi zwykłego telewizora, to zauważymy, że linie na brzegach ekranu są gorzej zogniskowane niż w jego centrum. Ktoś kto próbował kiedykolwiek regulować ogniskowanie linii wie, że nigdy nie udaje się uzyskać zadowalającego zogniskowania w centrum ekranu i na jego brzegach. W przypadku zwykłych odbiorników telewizyjnych nie ma to istotnego znaczenia, jednak w przypadku monitorów komputerowych, w których grubość kreślonej linii dochodzi do 0,2 mm występuje wyraźne pogorszenie jakości obrazu na brzegach ekranu. Aby temu zapobiec, podaje się na elektrodę ogniskującą kineskopu napięcie zależne od miejsca, w jakim aktualnie pada na ekran strumień elektronów. Dzięki temu uzyskuje się optymalne ogniskowanie strumienia w każdym punkcie ekranu.

W zależności od przeznaczenia monitora dobiera się odpowiedni luminator lampy kineskopowej. W przypadku przetwarzania tekstów najczęściej stosuje się luminofor o barwie bursztynowej. Ten typ luminoforu cechuje stosunkowo długi czas poświaty eliminujący migotanie obrazu powodujące u niektórych osób męczenie się wzroku. Nie nadaje się on jednak do wyświetlania ruchomych obrazów animowanych przez komputer, gdyż za poruszającymi się obiektami pozostają smugi psujące czytelność obrazu. Czasami jest to nieco męczące nawet w czasie przetwarzania tekstów, gdyż przesuwający się szybko tekst staje się nieczytelny. Ten typ luminoforu posiada dosyć niską żywotność i dłuższe pozostawienie na ekranie jakiegoś obrazu powoduje stosunkowo szybkie „wypalenie" luminoforu w jasnych jego miejscach, jednak z uwagi na niską cenę kineskopów (na świecie) ta wada nie ma istotnego znaczenia.

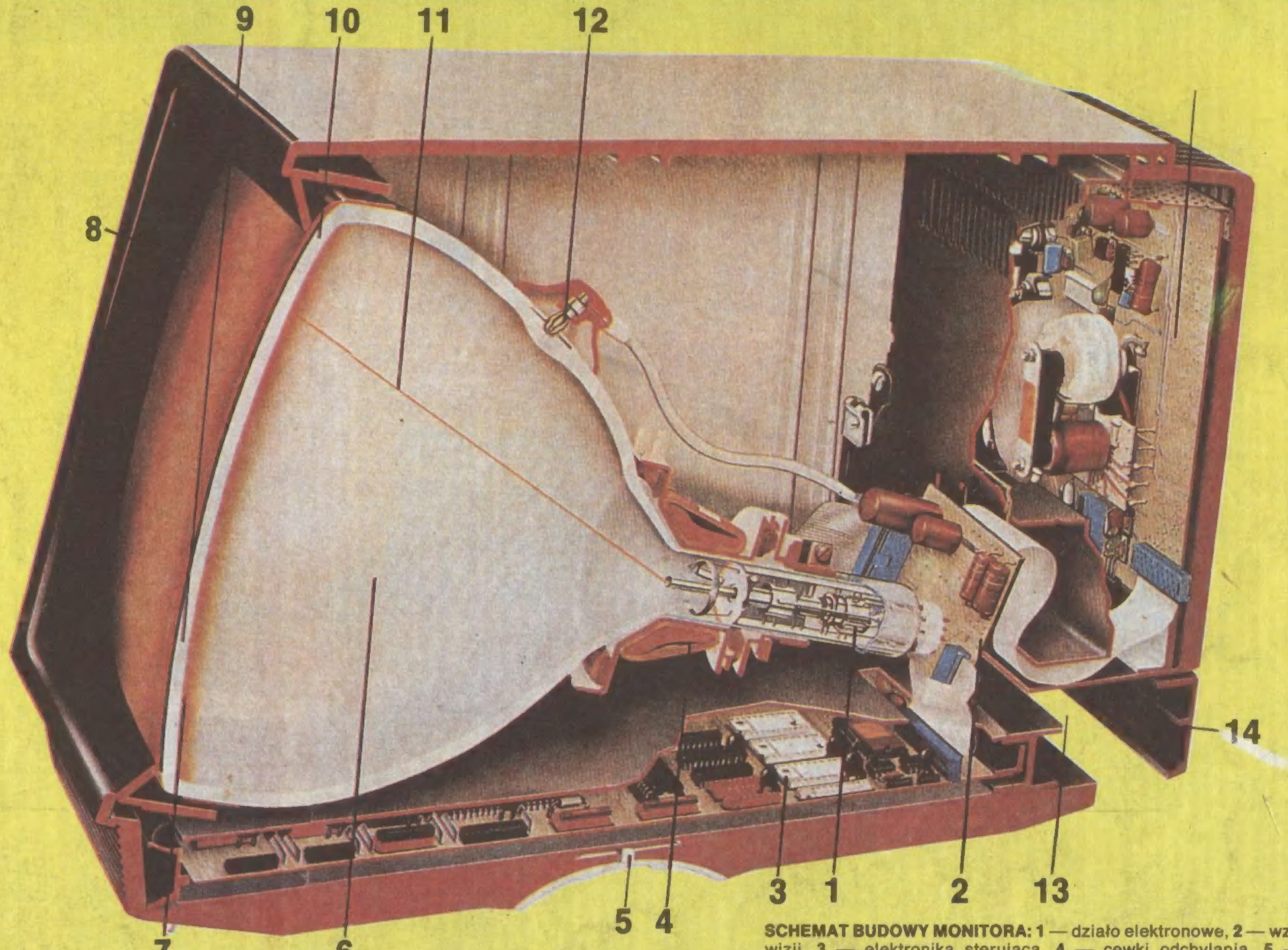
Monitor o zielonym kolorze świecenia jest również bardzo popularny. Jest on dobrze dostosowany do charakterystyki czułości widmowej oka i jego zastosowanie jest uniwersalne. Luminofor ten charakteryzuje się dużą żywotnością, co jest związane z jego dobrą wydajnością świetlną (pracuje z małymi prądami strumienia elektronów, co wpływa również na długowieczność katody), niektóre osoby nuży jednak zauważane przez nie migotanie ekranu, co jest z kolei związane z raczej krótkimi czasami poświaty tego typu luminoforów. Luminofory o białej barwie świecenia mają czas poświaty zbliżony do luminoforów zielonych, ich trwałość jest gorsza od trwałości luminoforów zielonych, jednak znacznie lepsza od trwałości luminoforów bursztynowych. Rzadko są one stosowane w monitorach komputerowych samodzielnie, częściej występują uzupełnione filtrem kolorowym „udając" monitory bursztynowe. Rozwiązanie to pozwala uniknąć niedogodności związanych z długim czasem poświaty monitorów bursztynowych, jednak przywraca efekt migotania oglądanego obrazu. Zastosowanie filtru poprawia jednak kontrast oglądanego obrazu — szczególnie w warunkach niekorzystnego oświetlenia zewnętrznego.

Monitory kolorowe dają bardzo efektowne zobrazowanie. Są one bardzo dobre w przypadku stosowania ich jako środków reklamowych lub jako terminali do niektórych dziedzin wspomagania komputerowego. Z uwagi na niewielką zazwyczaj rozdzielczość nie nadają się one do przetwarzania tekstów, gdyż powodują bardzo szybkie męczenie się wzroku. Należy zwrócić uwagę, że stosowane w tych monitorach luminofory (czerwony, zielony i niebieski) często posiadają inne charakterystyki spektralne niż luminofory stosowane w odbiornikach telewizji kolorowej, stąd stosowanie ich do współpracy z magnetowidami daje często zaskakujące efekty kolorystyczne. Wiele typów monitorów kolorowych posiada przełączniki pozwalające ustawić je w tryb pracy monochromatycznej, dając bursztynowy lub zielony kolor zobrazowania, nie poprawia to jednak ich rozdzielczości i prowadzenie przy ich użyciu obróbki tekstów jest bardzo męczące. Długowieczność lamp kineskopowych stosowanych w tych monitorach jest porównywalna z długowiecznością lamp o białej barwie świecenia ekranu.

Osobnego omówienia wymaga kwestia szkodliwego promieniowania emitowanego przez monitory. Producenci różnorodnych „ekranów ochronnych" przedstawiają je w swych ulotkach reklamowych często w kategoriach wręcz metafizycznych — kopiując zagraniczne rozwiązania często nie mają pojęcia o zasadach ich funkcjonowania. Prawda, jak zwykle bywa, nie jest jednak aż tak zagadkowa. Otóż w kineskopach kolorowych z uwagi na stosowanie różnorodnych masek pochłaniających znaczną część energii strumieni elektronowych, muszą dla uzyskania zadowalającej jaskrawości i rozdzielczości być stosowane wysokie napięcia anodowe. Napięcia te dochodzą do 30 kV. Elektrony hamowane na maskownicy i ekranie, część swej energii zamieniają wtedy na miękkie promieniowanie rentgenowskie. Promieniowanie to jest częściowo pochłaniane w szybie zewnętrznej wykonanej ze szkła ołowiowego, nie jest go jednak w stanie osłabić cienka siateczka z tworzywa sztucznego, nałożona na ekran. Występuje również inne zjawisko, ekrany monitorów kolorowych ulegają silnemu elektryzowaniu się, co powoduje, że część pyłków zawartych w powietrzu „odbija" się od nich i odskakuje z dużą prędkością; podobno przy zachowaniu zbyt małej odległości od ekranu pyłki te uderzają w oczy operatora, powodując szybkie męczenie się wzroku — tutaj siateczki dają pewną ochronę.

W monitorach czarno-białych, gdzie napięcia przyspieszające nie przekraczają zwykle 20 kV oba poprzednio opisane efekty nie występują (aby hamowany elektron mógł wypromieniować kwant promieniowania rentgena musi mieć dostarczoną energię). Siatka zastosowana jednak w dowolnym typie monitora poprawia kontrast uzyskiwanego obrazu. Mechanizm jej działania jest następujący: światło zewnętrzne przechodzi przez siatkę, odbija się od powierzchni luminoforu i przechodzi przez siatkę na zewnątrz — jest więc dwukrotnie tłumione przez siatkę, natomiast świecenie ekranu jest tłumione tylko podczas pojedynczego przechodzenia przez siatkę, co daje w efekcie poprawę kontrastu, jednakże okupioną zmniejszeniem jasności świecenia ekranu. Siatka nadaje też ekranowi specyficzną matową fakturę poprawiającą zobrazowanie znaków.

Piotr Postawka



SCHEMAT BUDOWY MONITORA: **1** — działo elektronowe, **2** — wzmacniacz wizji, **3** — elektronika sterująca, **4** — cewki odchylania, **5** — uchwyt mocujący, **6** — anoda przyspieszająca, **7** — luminofor, **8** — maskownica, **9** — szyba przednia, **10** — filtr kolorowy, **11** — strumień elektronów, **12** — kontakt anody, **13** — miejsce na przewód zasilania, **14** — uchwyt transportowy

Cena zł 200,—

Indeks nr 366013 PL ISSN-0860-5696